Module 4: How Do I Preprocess Data?

Table of Contents

Lesson 4-0: Module 4 Overview

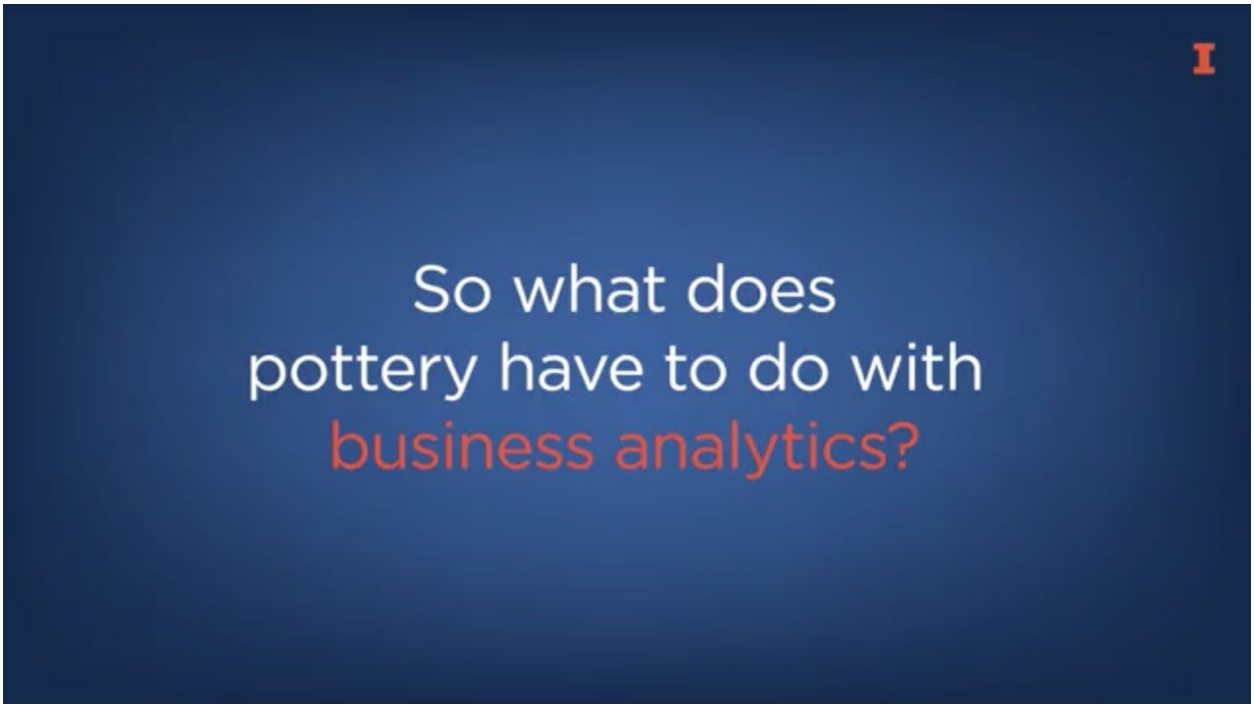
Lesson 4-0.1 Introduction

Converting raw clay into a beautiful vase that has both aesthetic and utilitarian value, requires several steps. After the clay has been filtered and cleaned to remove any rocks or other impurities, the potter uses tools to reshape a slab of clay into a vase like form

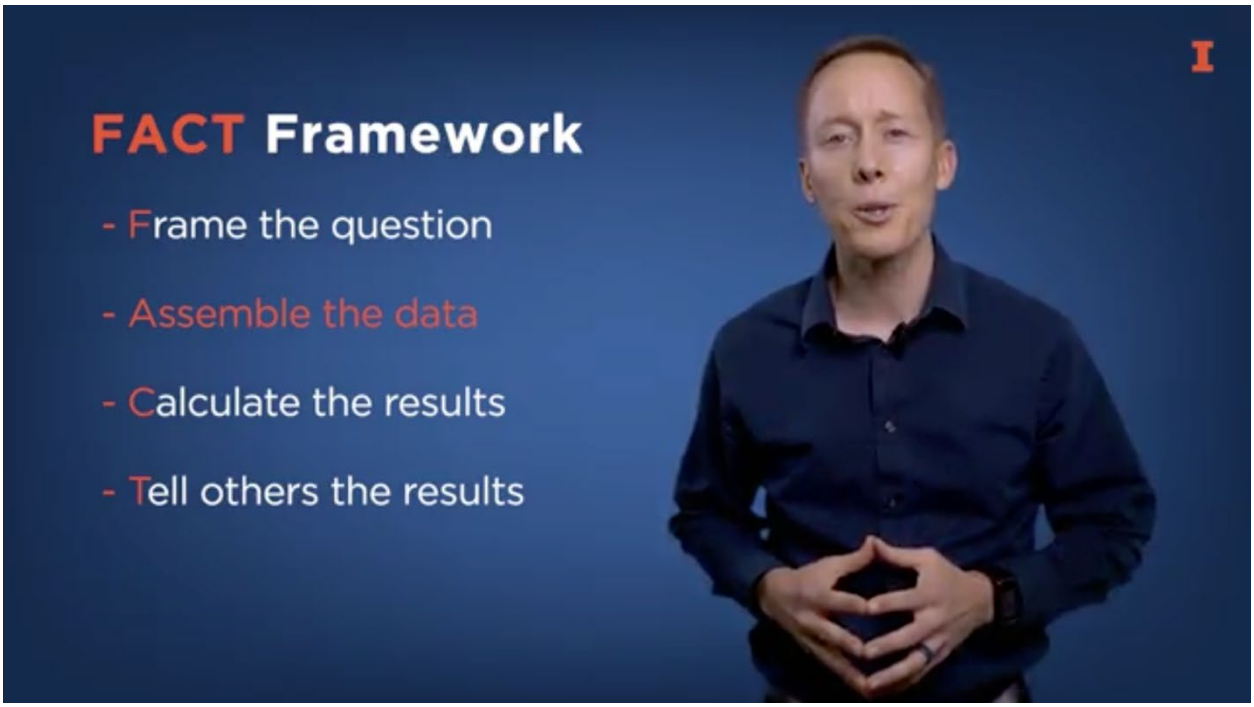
before it is glazed and then put into a kiln and fired. The end goal is to turn the slab of clay into a product that is both beautiful and useful.

The process that goes into reshaping a slab of clay into a vase is of particular relevance to this module.

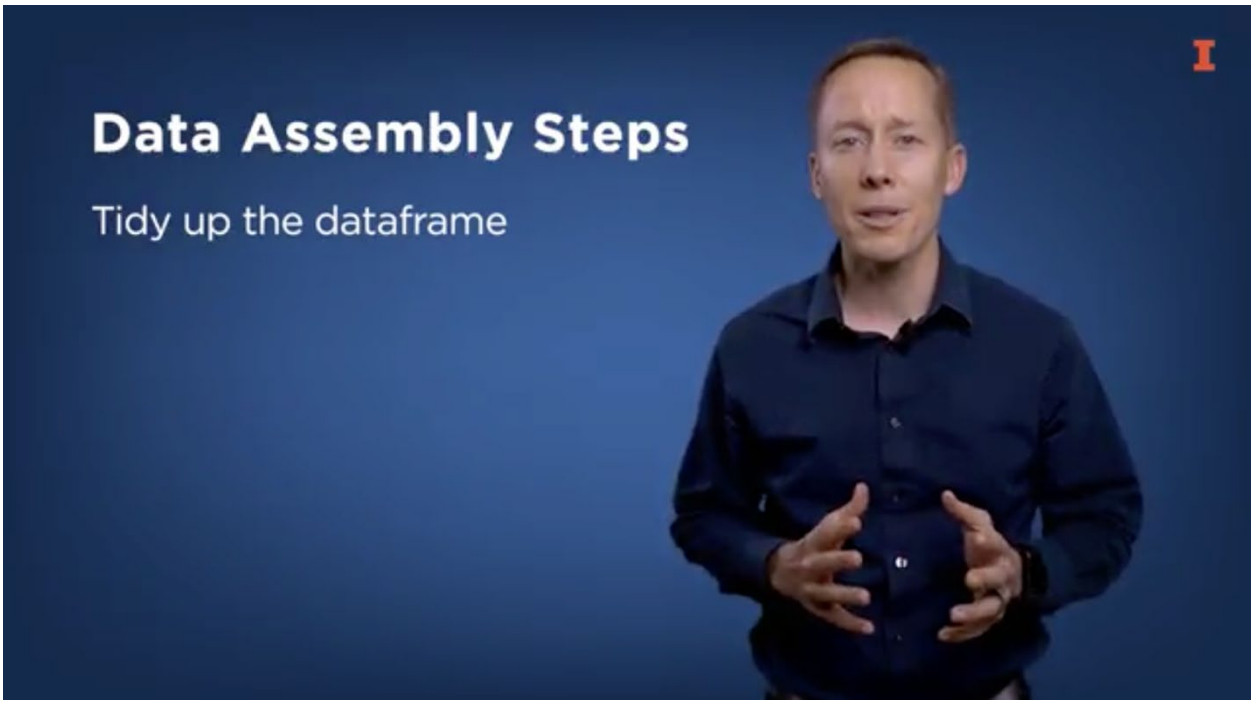
In this video, notice how the potter is using a pottery wheel to help in this process. Also notice how the potter moves their hands up and down the clay repetitively. This massaging of the clay makes the clay taller and the walls thinner. The potter can also make it shorter and thicker by pushing down on it.



What does pottery has to do with business analytics?



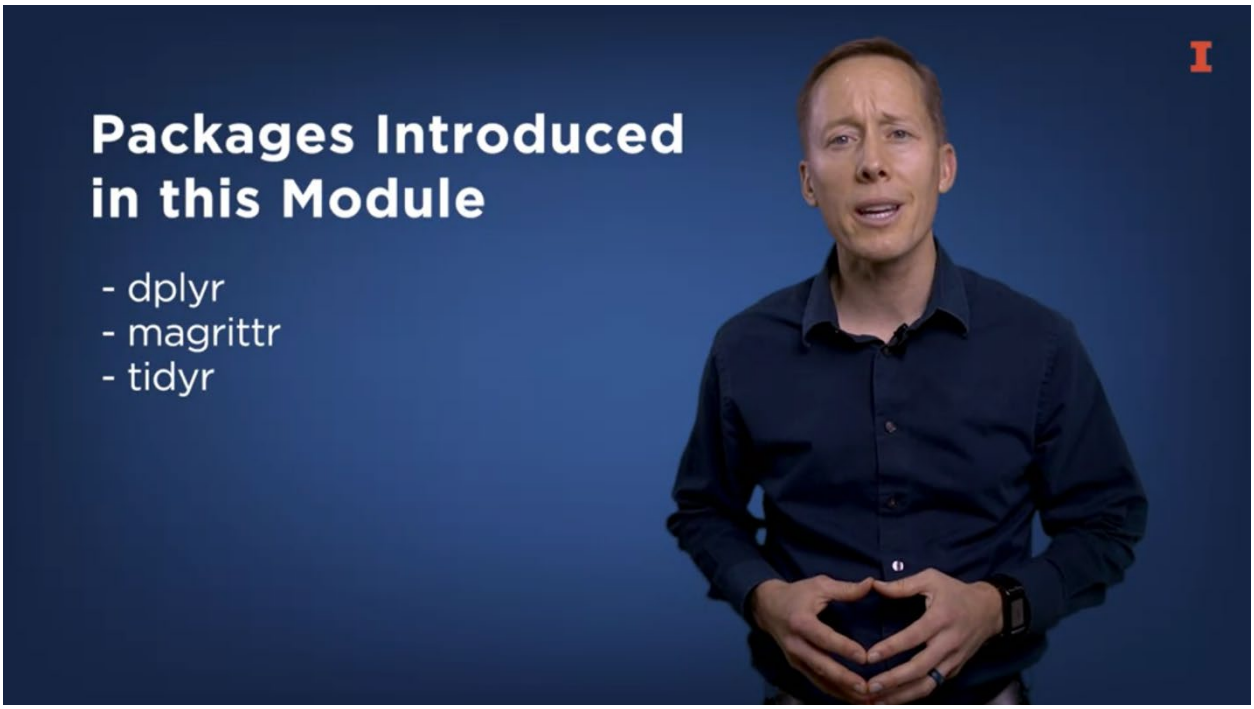
The A in the fact framework refers to steps required to assemble data.



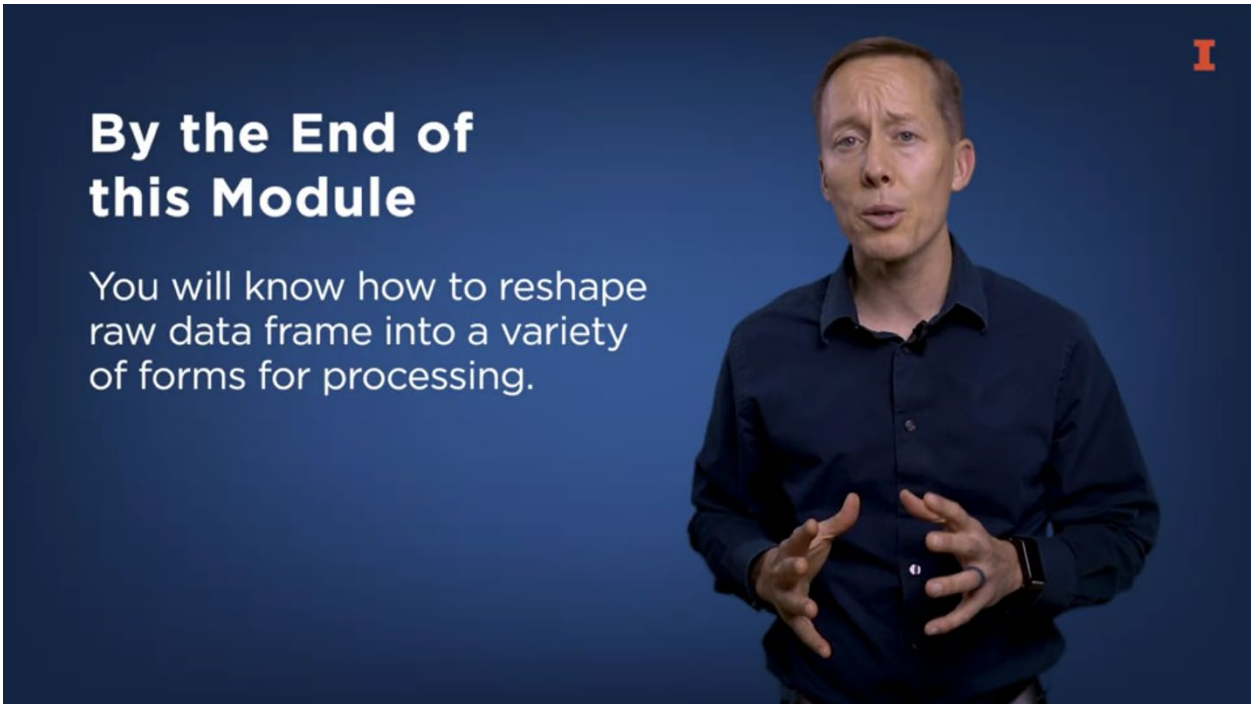
Tidying up the data is similar to removing the impurities from the clay.

After tidying up the data, additional pre-processing is required to prepare the data for analytic tools. Specifically, the data needs to be aggregated and reshaped in such a way that it can fit into a data analytic tool like visualization software or an algorithm. Just as a potter relies heavily on a pottery wheel to help shape the clay, there are some often used tools that a data analyst uses to aggregate the data and shape the data frame.

In this module, you'll become familiar with several factors that influence how the data is pre-processed before it can be processed to gain actionable insight. Specifically, we will analyze how the business question, the managerial responsibility, the level of granularity of the raw data, and the trade-off between operational control and operational feasibility influence the level at which data is aggregated. We will then introduce the idea of pivoting a data frame between wide and long shapes so that it will fit into the data analytic tool that processes the data for actionable insights.



You will then get experience aggregating and reshaping data using several functions from the dplyr, magrittr, and tidyr packages.



The hope is that by the end of this module, you will know how to take a tidy but raw data frame and reshape it into a variety of forms so that it can fit into the desired analytic tool, and then be processed to obtain actionable insight.

References

Shkraba, A.(2020). Video of man doing the art of pottery[Video]. CC0 1.0. Pexels.

bluebudgie.(2017). vase-2848243[Online image]. CC0 1.0. Pixabay.

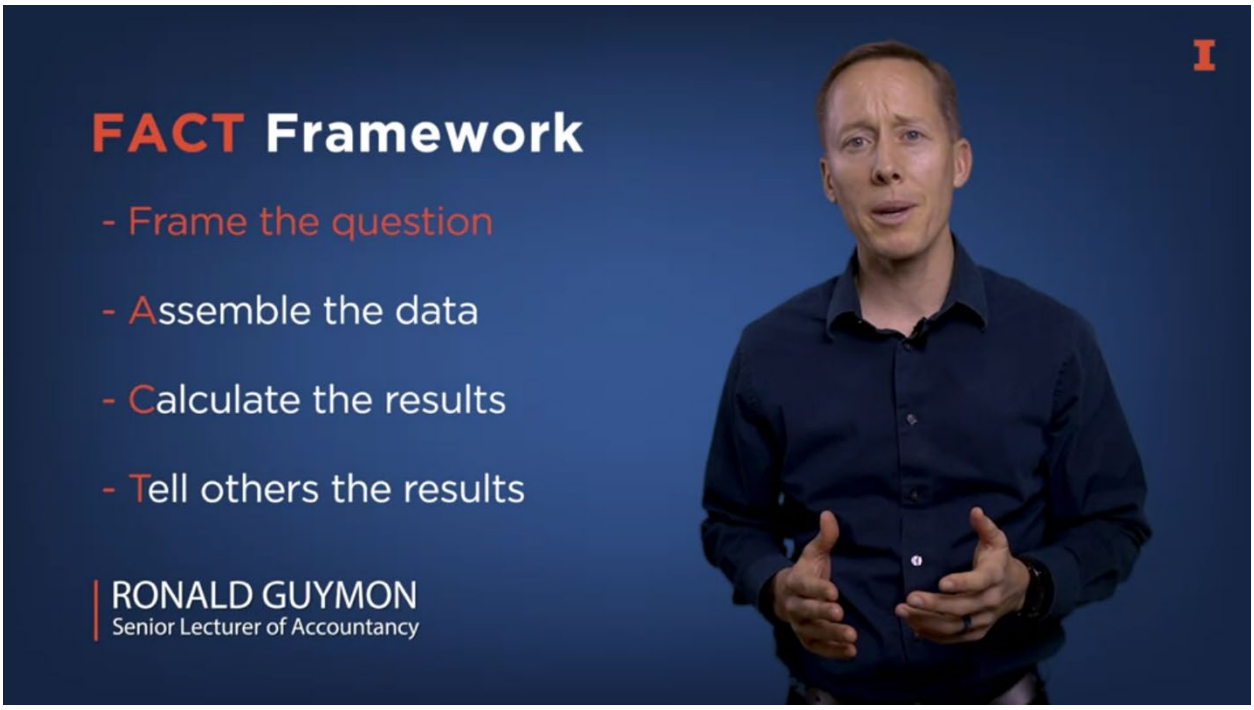
Shkraba, A.(2020). Video of personworking on pottery business[Video]. CC0 1.0. Pexels.

Lesson 4-1: Lectures

Lesson 4-1.1 Introduction

Welcome to Module 4. In this module, we will learn about data visualization techniques and R for exploratory data analysis. Data visualization may serve two purposes, data exploration for patterns in the raw data and communication of the results of analysis. The purpose of this module will be data exploration. The communication aspect will be covered in the next course in this specialization. In this module we will use another popular tidyverse package called ggplot2. In fact, you will see that ggplot2 is much more than a visualization package. It's a data visualization philosophy. I look forward to seeing you in the next video.

Lesson 4-1.2 Framing Questions for Actionable Insight



The first and most important step in the Fact Framework is to frame a question in a way that can be answered using data analytic tools. The question should also lead to an answer that can be acted upon to further the organization's goals. Let's take a minute to think about the importance of framing your question in the context of American football.

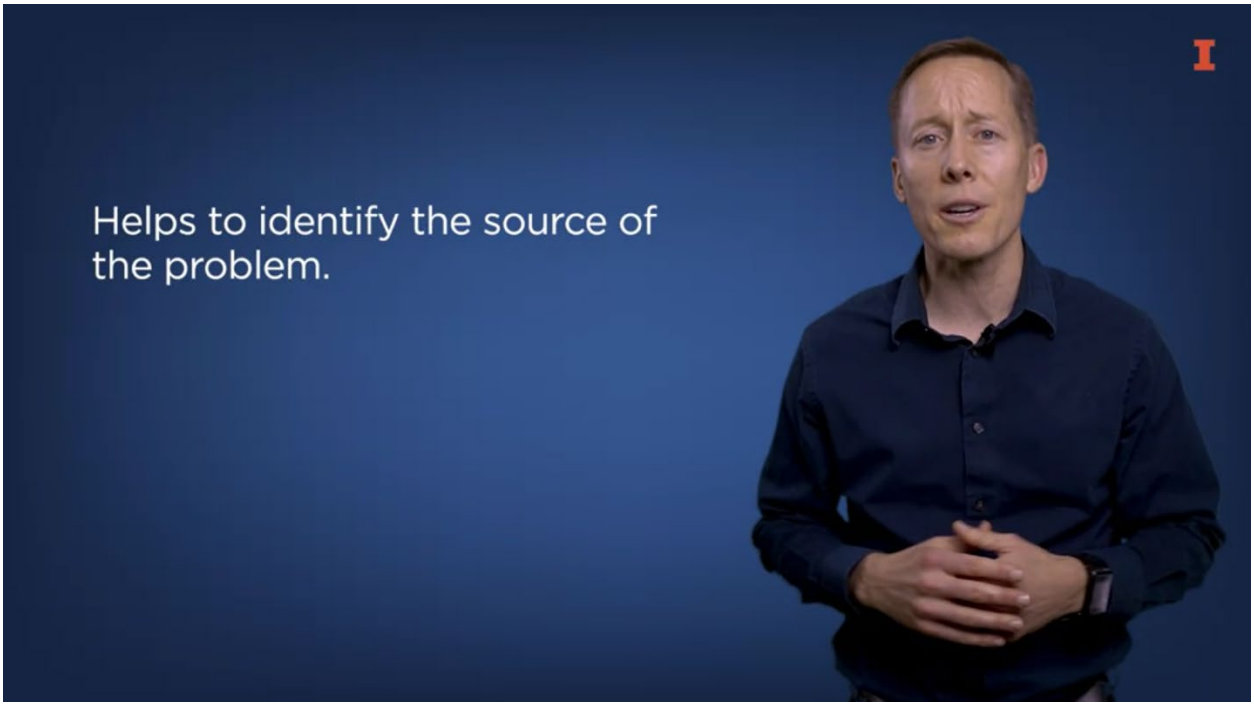
Every spring, the National Football League has a draft in which teams get to choose which college athletes they will hire onto their team. In my mind, this seems like a very complex decision because American football teams have many different positions. I asked Luis Guillermo, Director of Analytics and Applications for the NFL's Buffalo Bills American football team, how they approached the NFL draft. Let's listen to his answer and consider how he frames the question.

Though when approaching the question of the draft, like any big question, you want to break it down into smaller pieces. The first step to doing that is understanding the field. When I say the field is now what prospects are available to be drafted and then understanding each one of those players as well as possible. That process is a very long process, I mean, you really focus on it for about six solid months, but that starts long before that. The second piece to the question is really understanding your own roster as well. Most teams are really good about understanding their deficiencies of their roster,what are their strong points? Where you really need to get better, where you want to get better, but at the end of the day, you just want to get better everywhere. Then finally is that you want to understand everybody else's roster as well, because then you can understand what their needs are, where they're going to be looking to be drafting what they might be trying to do, and utilizing those three components, you can put them together to figure out which direction you want to go. The main thing is that just because you need a position, it doesn't mean that you have to draft that position, and just because you're strong at a position, it doesn't mean that you don't want to draft that position. The goal is to get stronger everywhere, and anywhere, and if you end up with additional surpluses, there's other avenues of getting players through trades and whatnot, through free agency, of course, and and waivers.

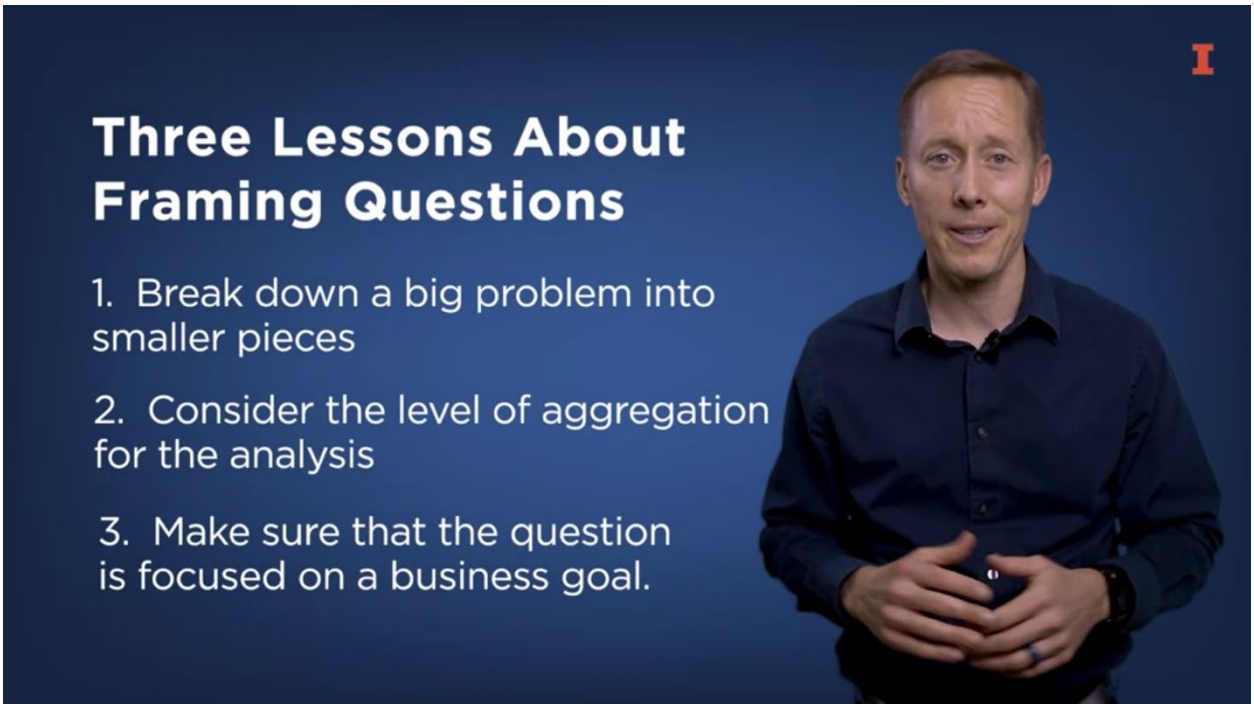
I think there are many lessons to learn from the Luis' response, but I want to focus on just three of them. First, he said that a big problem is broken down into smaller pieces.

Then he described the three pieces; one, understanding the field, meaning the available players, two understanding your own roster or the players who are currently on your team, and three, understanding everybody else's roster or the players who are on other teams.

Breaking a problem down into smaller pieces is especially consequential in a business setting because it speeds up the analysis by allowing multiple people to work in parallel on gathering and analyzing data.



It's also helpful because it shows some level of thought about the source of the problem. Notice that Luis' third question explicitly acknowledges, that they need to consider the skills of their opponents.



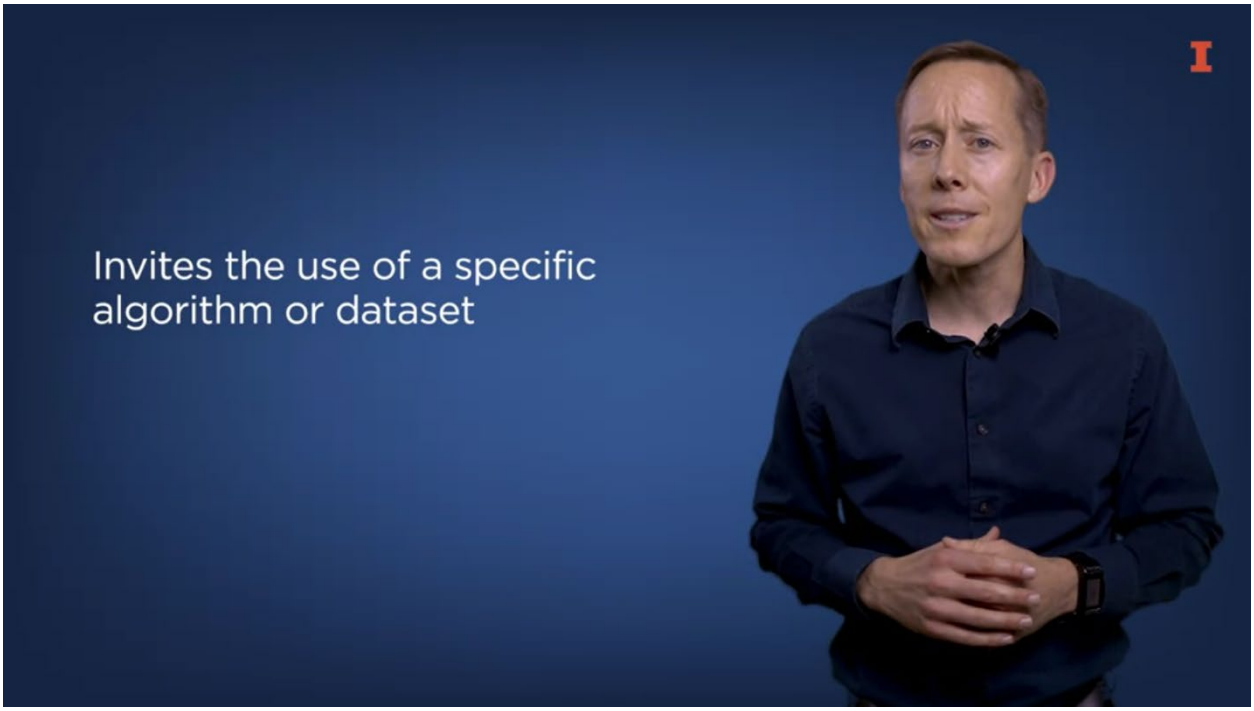
A second lesson from Luis' response relates to the level of aggregation of the analysis. Notice that each of the three pieces of the overall question are focused on the player level rather than say, at the level of a play or the game, or a team, or the season level. Now, it may seem obvious that the data should be aggregated at the player level if you want to answer questions about the player, even though the data most likely was not originally aggregated at that level. The third lesson I want to expand upon is that the main goal for Luis' organization during the draft is not to get the players who best fit the current needs of their team, but to get the best players, period.

Thus, the question for the draft could be framed as, how can we identify the best players? Luis mentioned that there are other avenues the NFL can use to get the players they need after the draft is over, such as trades and waivers. Now that we've identified a question that focuses more specifically on the problem, we want to further refine that question so that it encourages a careful methodical approach that will lead to actionable insight.

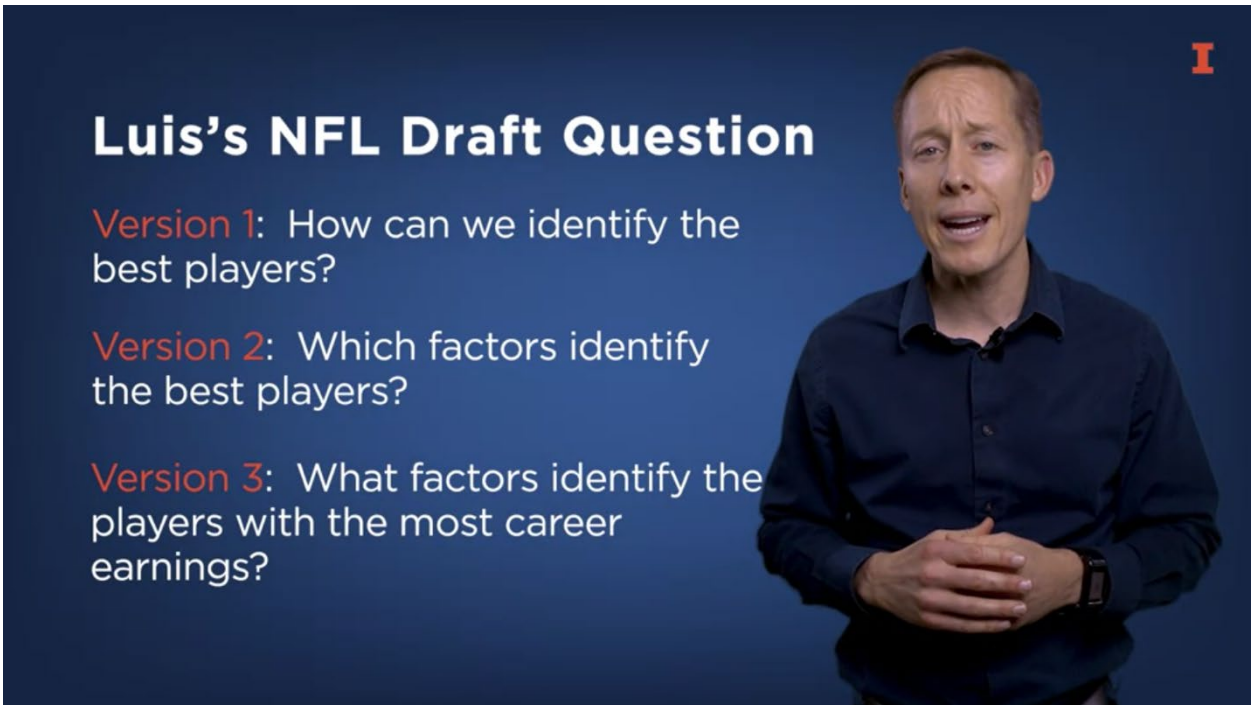
This type of refinement will take curiosity as well as a deeper understanding of data and how to analyze it.

For instance, Luis could refine the question to ask, which factors identify the best players? This question is only slightly different from the previous question, but the difference is important because it invites the analyst to be specific and explaining what causes or at least is associated with great players. This would require the analyst to not

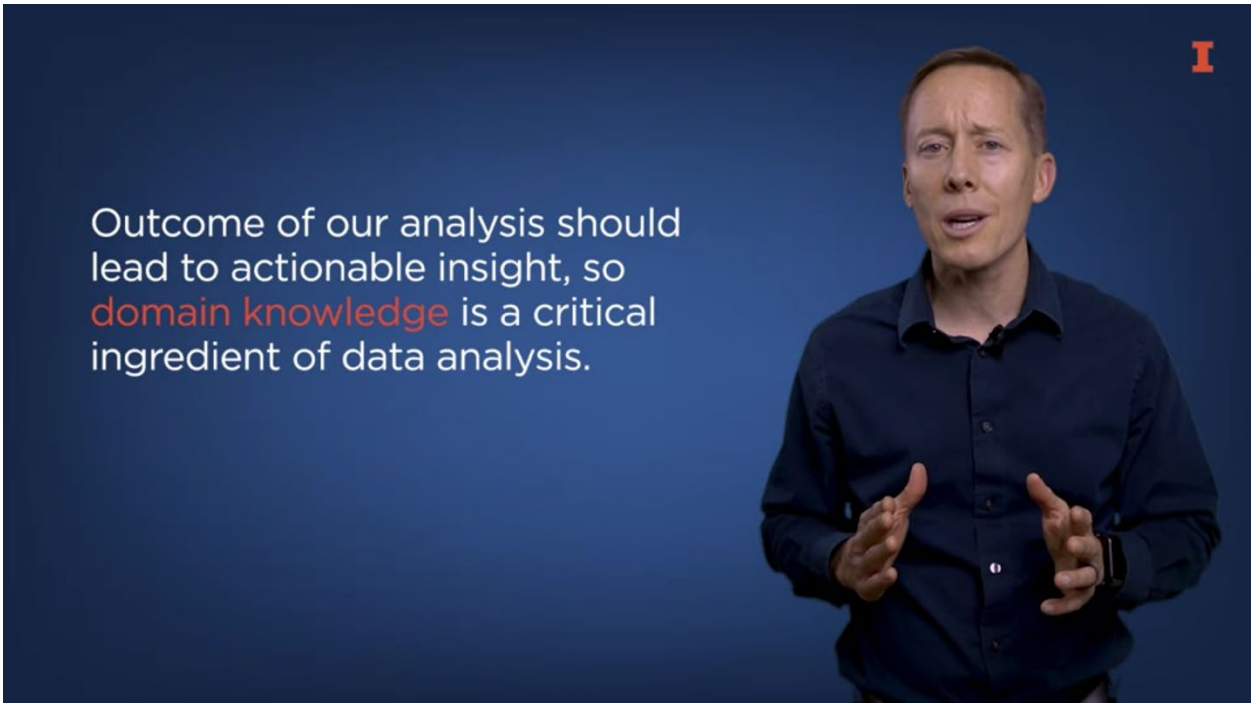
only identify great players but also to provide some insight about the characteristics that are most important in a player.



As you learn more about different data analytic algorithms, you may want to consider how to ask the question in a way that invites the use of a specific algorithm or dataset. For instance, an additional refinement of the question could be, what factors identify the players with the most career earnings?



This question clarifies what is meant by best. It is more specific, so it would require the analysts use data about player's career earnings, which was not implied by the prior question.

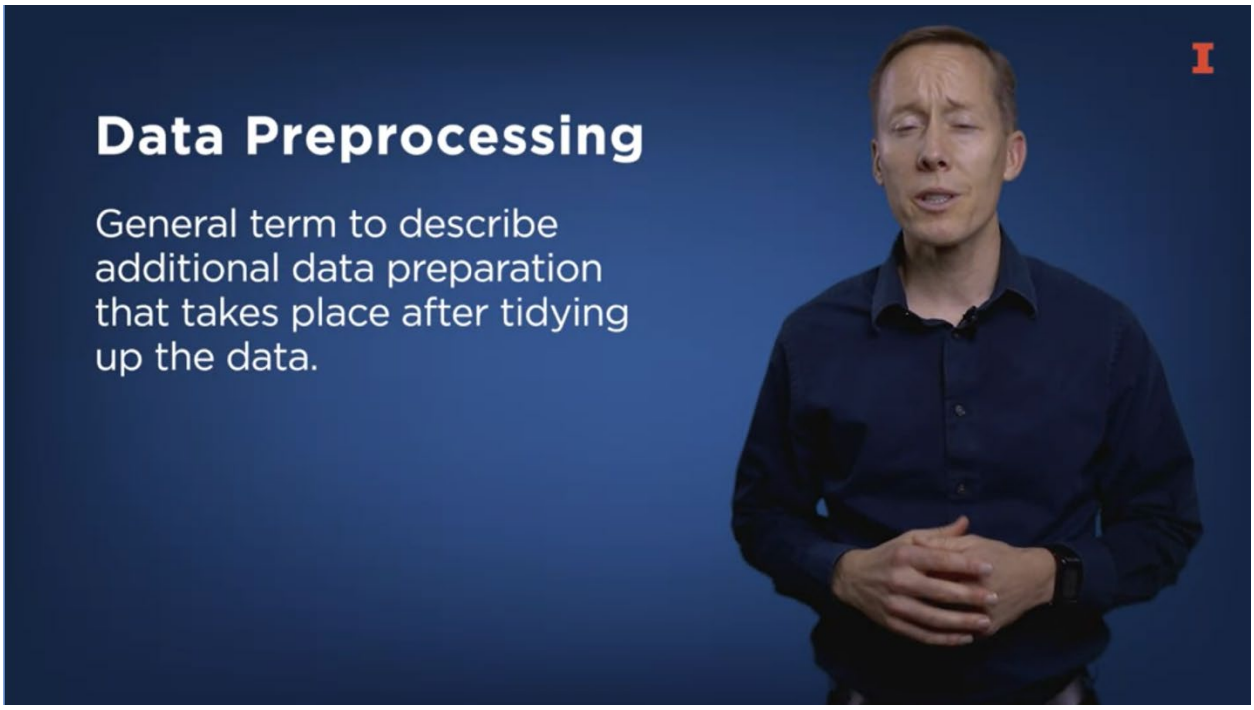


Now, remember that the outcome of our analysis should lead to actionable insight. Domain knowledge is a critical ingredient of data analysis. In terms of our NFL example, we would need to consider details related to how the draft works. It may be the case

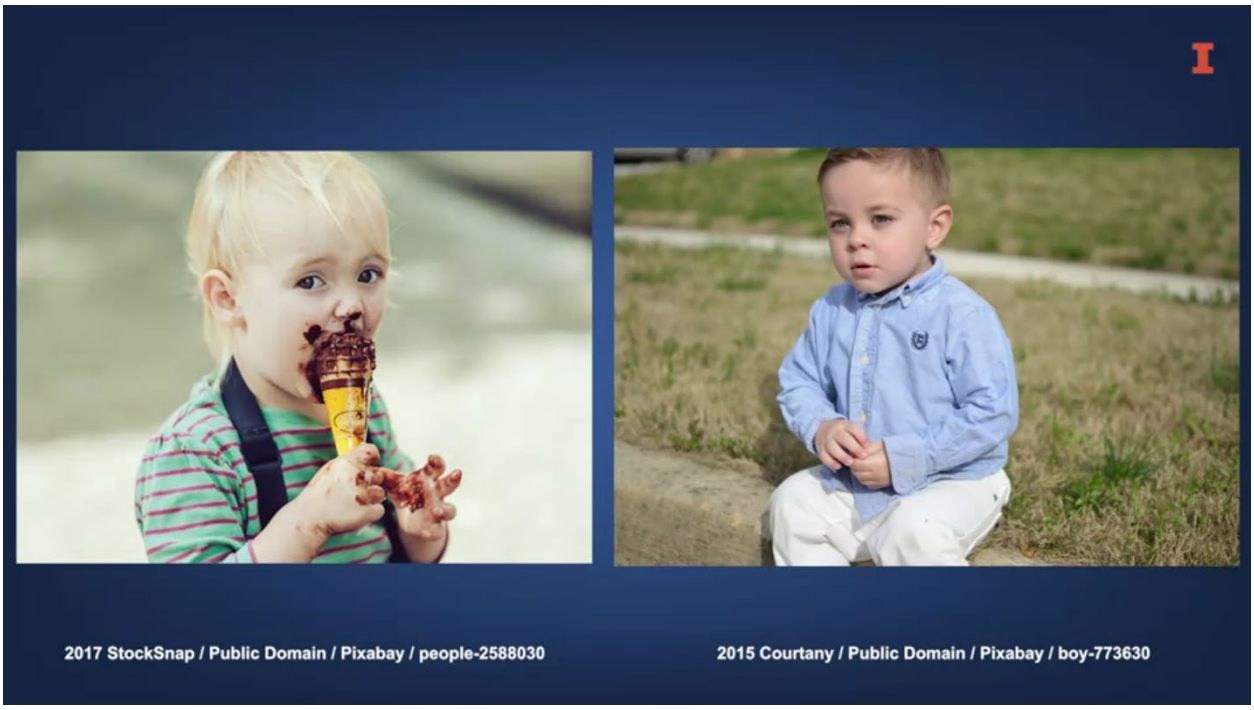
that we want to simply create a rank ordering of the players and pick them based on the highest ranked player available for each turn.

If that's the case, then we may want to refine the question to, what factors lead to a rank ordering of players based on expected career earnings? In conclusion, I want to emphasize that your ability to frame a question is important because it will affect the approach that you take in data analytics. If a careful methodical approach is followed, then the answers are more likely to be trusted and then acted upon.

Lesson 4-1.3 Dataframe Shape: Level of Aggregation

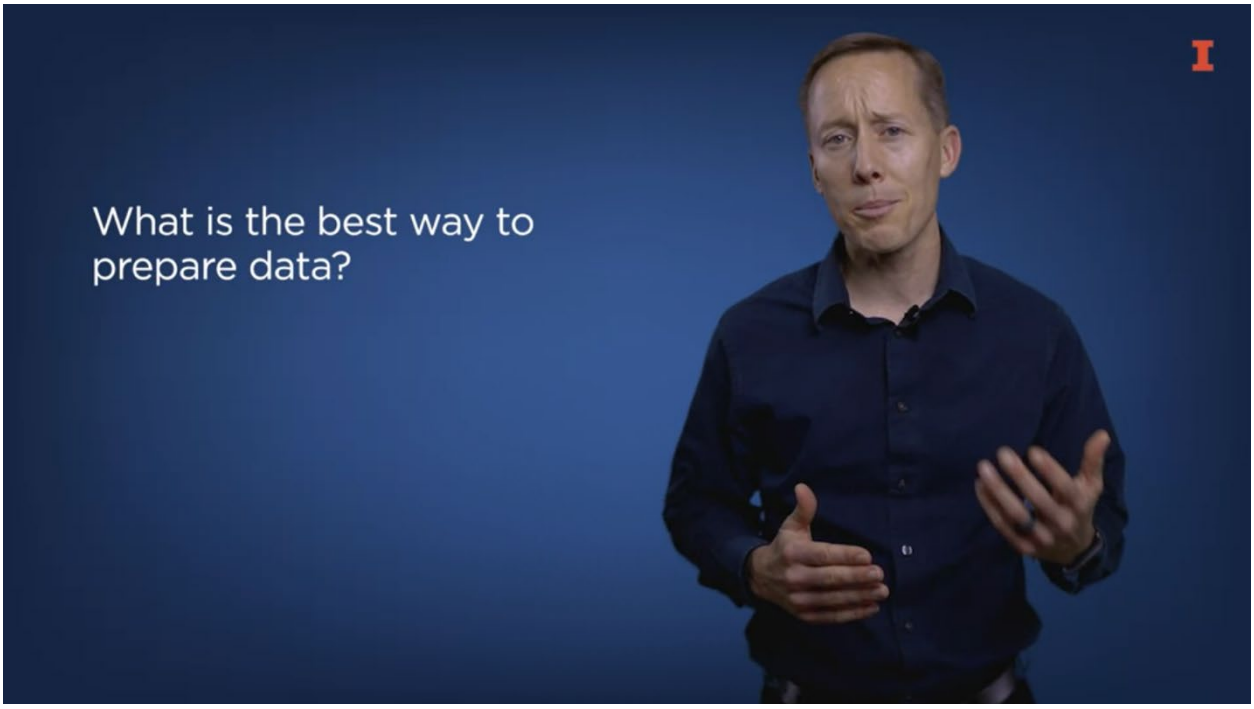


This lesson focuses on the level at which data should be aggregated, which is just one aspect of preprocessing data. Preprocessing data refers to additional data preparation after tidying up the data and is one aspect of assembling data.



Oftentimes, raw data looks more like the child on the left, while the data that goes for

the modeling looks more like the child on the right. The point is that we need to tidy up the data before we analyze it. Once the data is tidy, we need to preprocess the data or prepare it to fit the business question.



So what is the best way to prepare data? It's a lot like asking how to dress for an event.

For instance, in which of these images does the child look more prepared for the party, the one on the left, or the one on the right? Well, you may have already guessed that the answer depends on which party the child is going to attend. If it's a Halloween party, the child on the left is more prepared than the child on the right. On the other hand, if it's a birthday party, the child on the right is more prepared than the child on the left.

With respect to preparing data in business analytics, there are three key factors that contribute to the level of data aggregation. The first factor is the manager's decision domain.

For instance, if a manager is responsible for the sales team, then the manager may often want to aggregate data at the level of individual sales agents. In other words, each

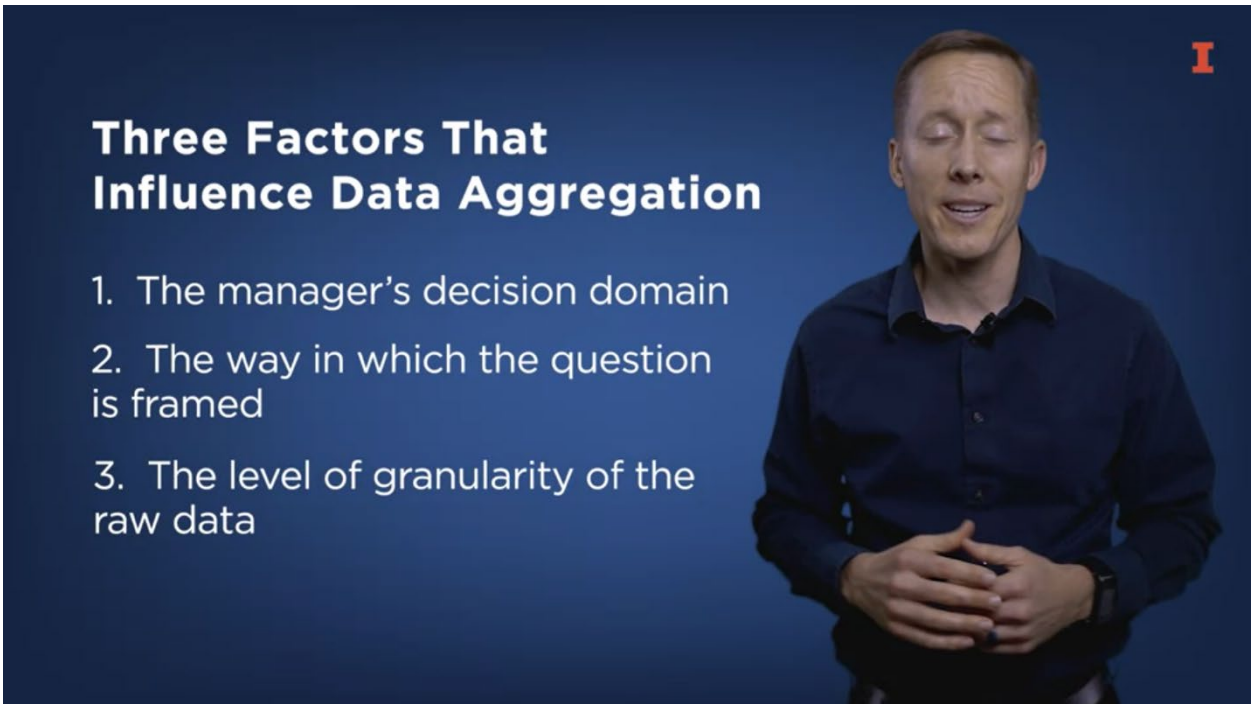
row should represent an individual sales agent and each column should represent a characteristic about that agent.

On the other hand, if a manager is responsible for manufacturing a category of product lines, then the manager may want to aggregate data at the level of individual products. In this case, each row should represent a product and each column should represent a characteristic of the product.

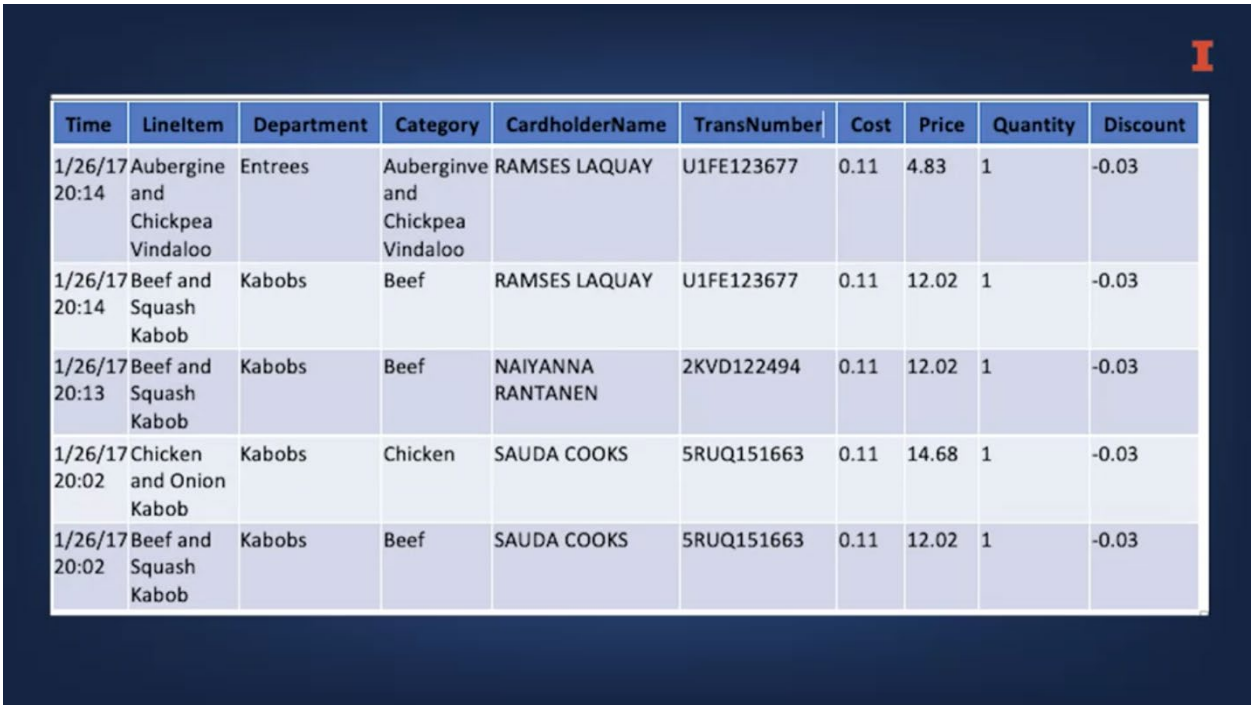
The second factor that contributes to how the data is prepared is the way in which the question is framed.

Quarter Ending	Quantity Sold	Sales Revenue	Variable Costs	Fixed Costs	Profit
Mar 31, 2021	10,050	603,000	201,000	50,000	352,000
Jun 30, 2021	12,250	735,000	245,000	52,000	438,000
Sep 30, 2021	14,500	870,000	290,000	51,000	529,000
Dec 31, 2021	21,650	1,299,000	433,000	52,500	813,500

Specifically, if the sales manager frames a question that analyzes individual sales agents over the prior 12 quarters, then the data should be aggregated such that each row represents a sales agent's performance for a specific quarter.

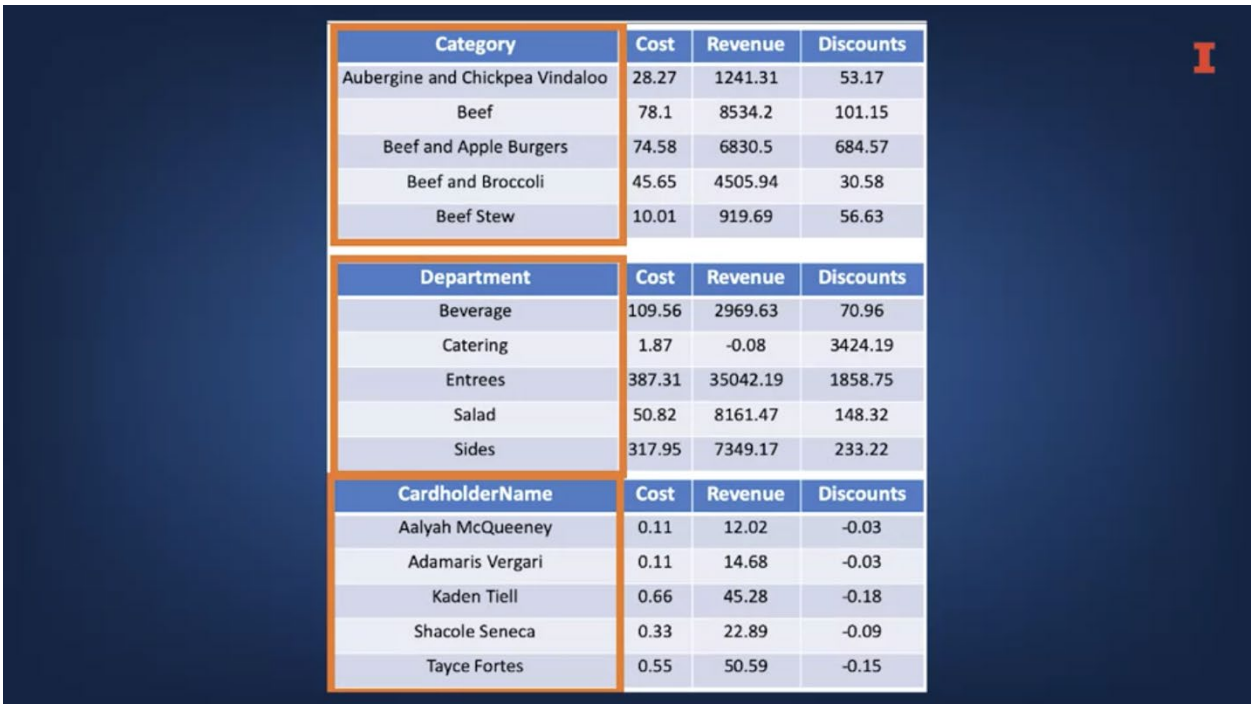


The third factor that contributes to how data is prepared is the level of granularity of the raw data.



Time	LineItem	Department	Category	CardholderName	TransNumber	Cost	Price	Quantity	Discount
1/26/17 20:14	Aubergine and Chickpea Vindaloo	Entrees	Auberginve and Chickpea Vindaloo	RAMSES LAQUAY	U1FE123677	0.11	4.83	1	-0.03
1/26/17 20:14	Beef and Squash Kabob	Kabobs	Beef	RAMSES LAQUAY	U1FE123677	0.11	12.02	1	-0.03
1/26/17 20:13	Beef and Squash Kabob	Kabobs	Beef	NAIYANNA RANTANEN	2KVD122494	0.11	12.02	1	-0.03
1/26/17 20:02	Chicken and Onion Kabob	Kabobs	Chicken	SAUDA COOKS	5RUQ151663	0.11	14.68	1	-0.03
1/26/17 20:02	Beef and Squash Kabob	Kabobs	Beef	SAUDA COOKS	5RUQ151663	0.11	12.02	1	-0.03

If, for instance, the raw data is point of sale data that has a separate row for every item in a transaction and includes columns for time, line item name, department, category, cardholder name, transaction number, cost, price, quantity and discount, then it's very granular. And you have a lot of flexibility in aggregating the data at a variety of levels.

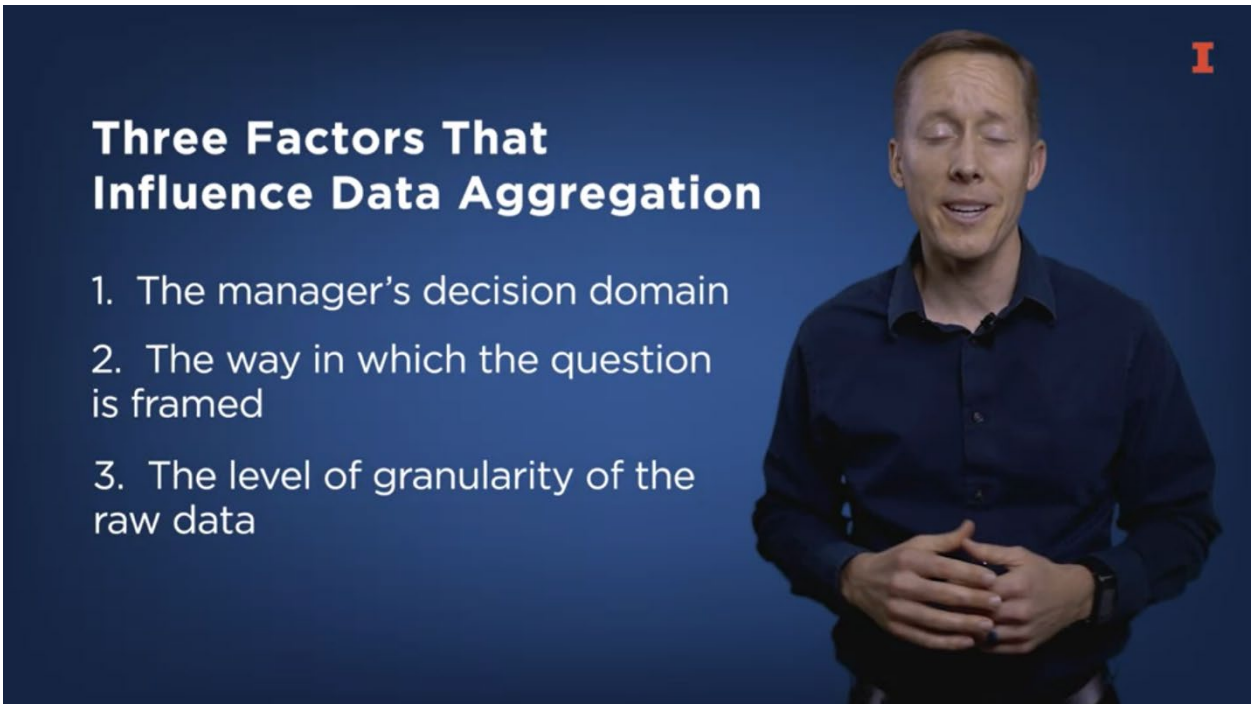


Category	Cost	Revenue	Discounts
Aubergine and Chickpea Vindaloo	28.27	1241.31	53.17
Beef	78.1	8534.2	101.15
Beef and Apple Burgers	74.58	6830.5	684.57
Beef and Broccoli	45.65	4505.94	30.58
Beef Stew	10.01	919.69	56.63

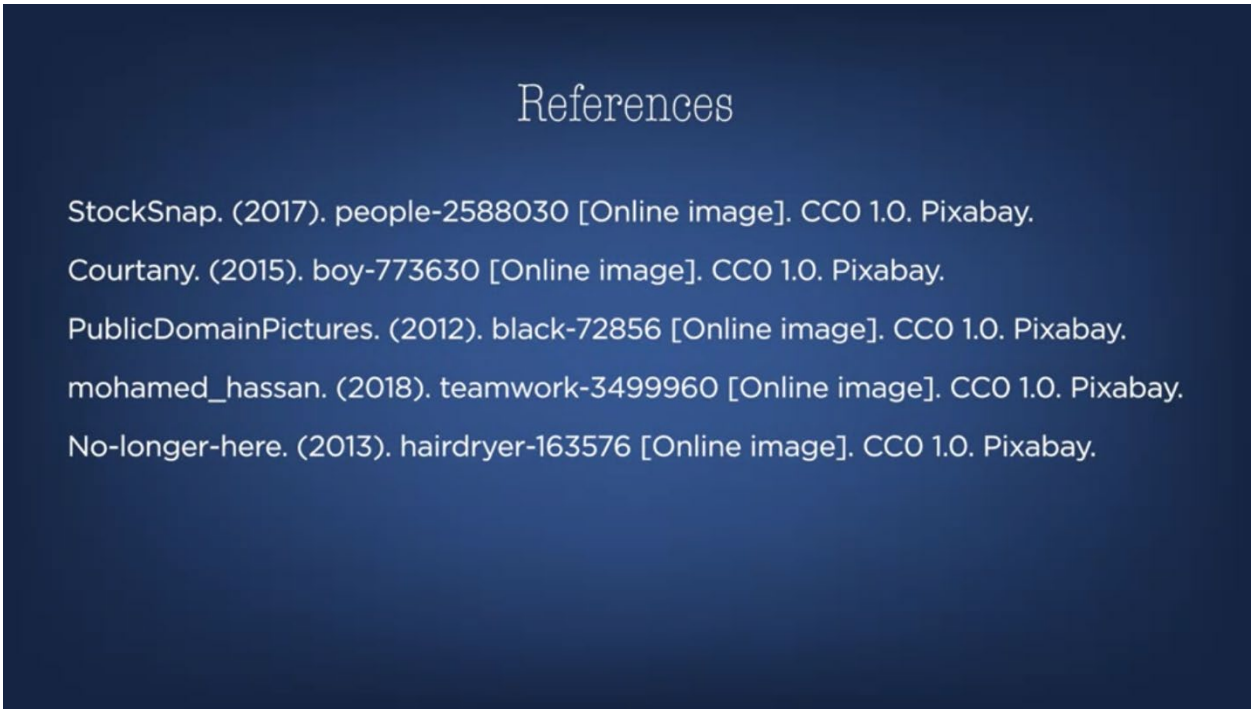
Department	Cost	Revenue	Discounts
Beverage	109.56	2969.63	70.96
Catering	1.87	-0.08	3424.19
Entrees	387.31	35042.19	1858.75
Salad	50.82	8161.47	148.32
Sides	317.95	7349.17	233.22

CardholderName	Cost	Revenue	Discounts
Aalyah McQueeney	0.11	12.02	-0.03
Adamaris Vergari	0.11	14.68	-0.03
Kaden Tiell	0.66	45.28	-0.18
Shacole Seneca	0.33	22.89	-0.09
Tayce Fortes	0.55	50.59	-0.15

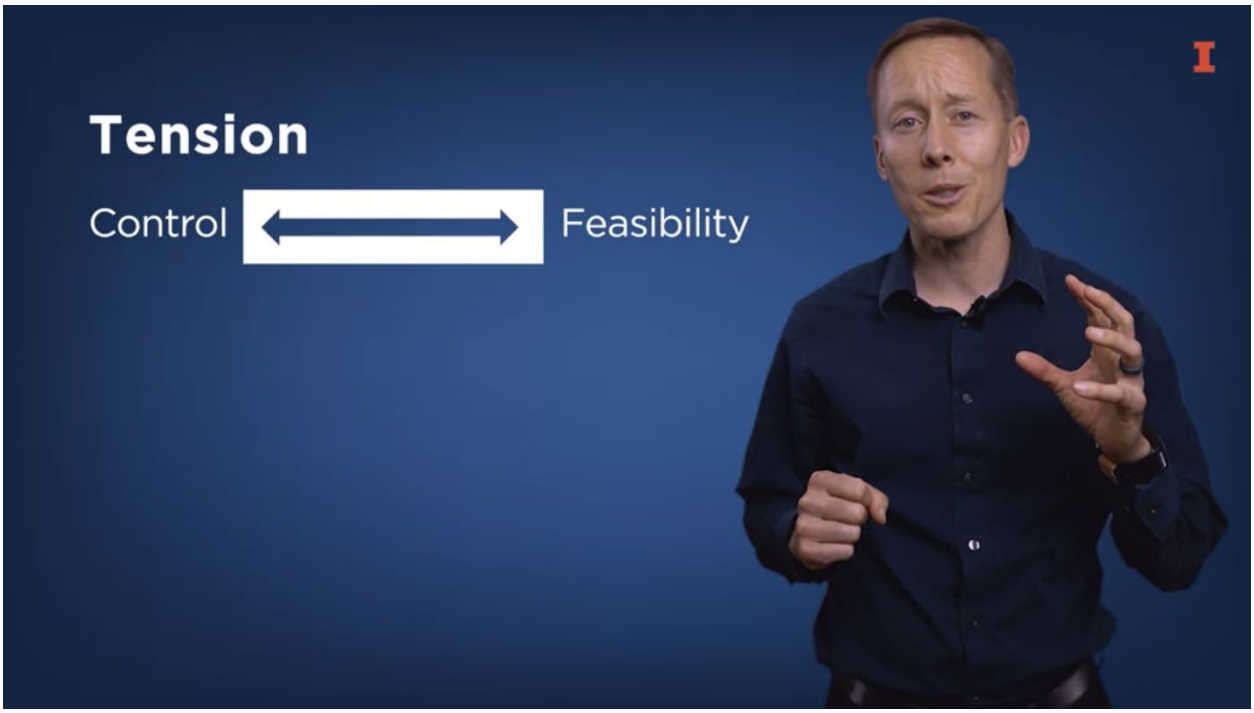
For instance, you can aggregate the data at the category department or customer level. On the other hand, if the raw data comes in an aggregated format, then you can't disaggregate it to a lower level. For example, if the sales data comes in a way such that each row represents sales for the department during the day, then it would be impossible to disaggregate it down to the customer or line item level.



In conclusion, the level at which data is aggregated is an important aspect of preprocessing data for analysis. And the level of aggregation depends on the manager's domain of responsibility, the question frame and the level of granularity of the raw data.

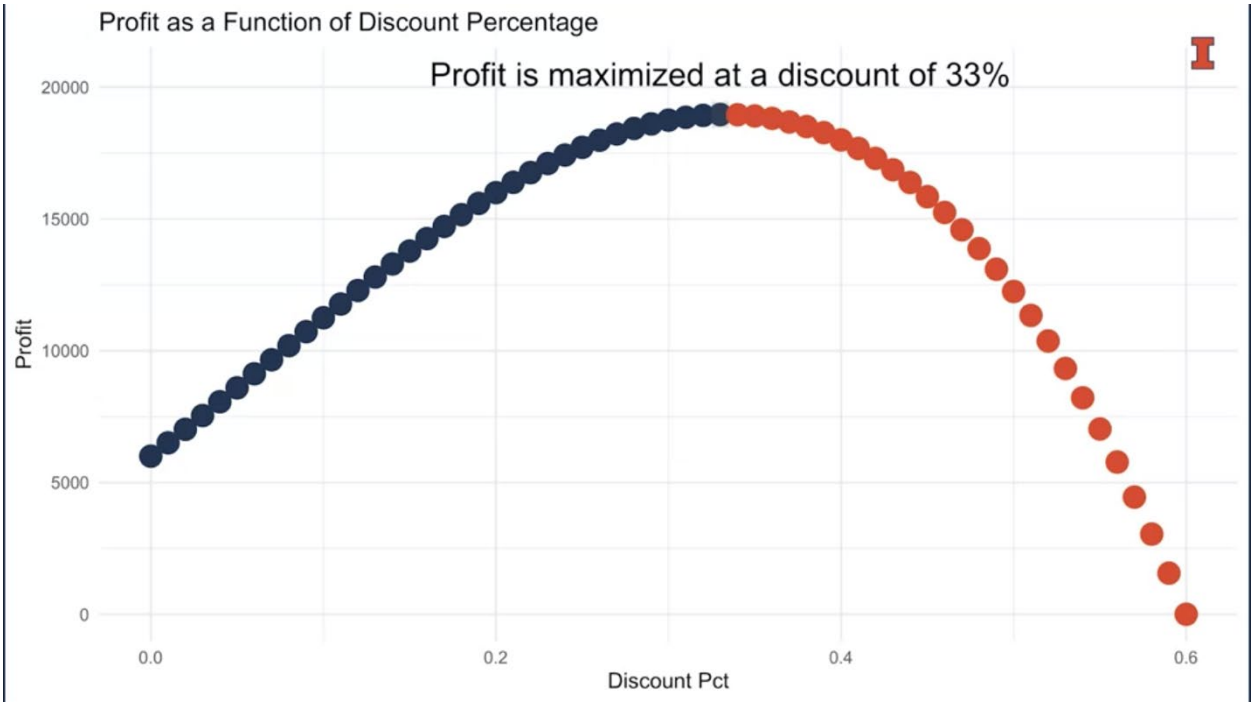


Lesson 4-1.4 Dataframe: Control Versus Feasibility



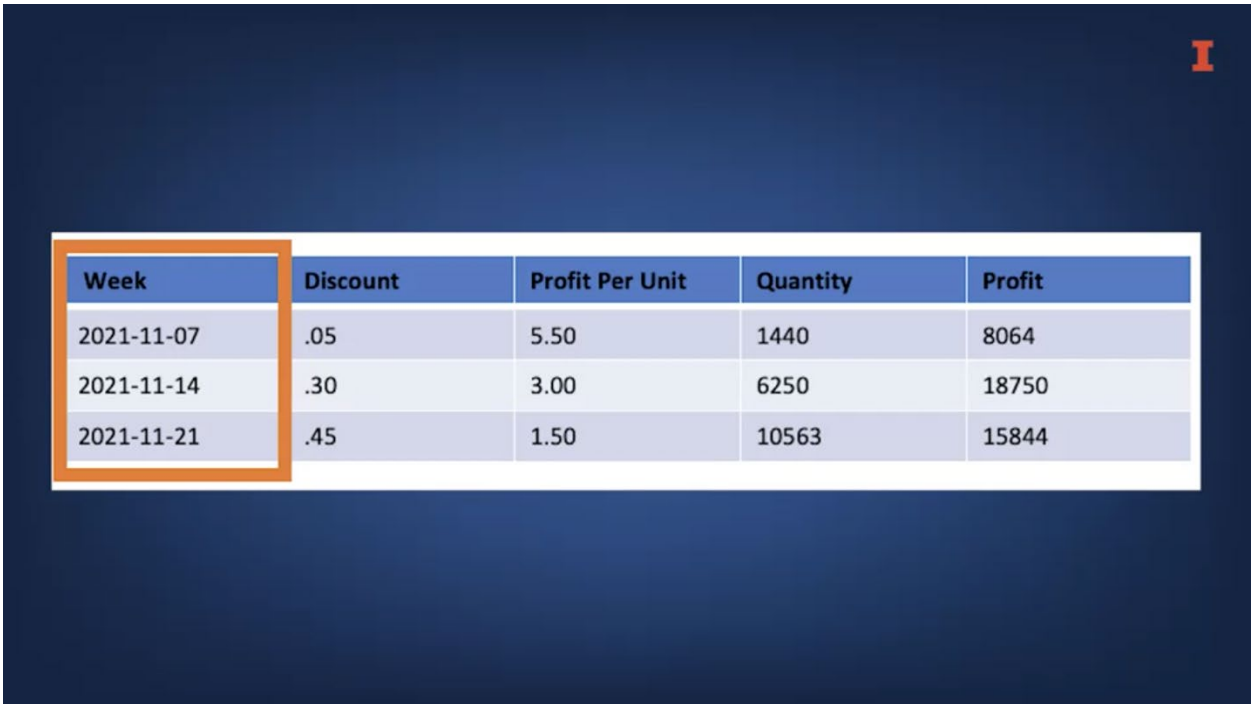
In this lesson, we'll focus on an issue related to the level at which data is aggregated, the tension between control and operational feasibility. The reason why there is tension between these two concepts is because a low amount of control is operationally very feasible but it is not desirable. While a high amount of control is operationally infeasible but is desirable.

The whole idea of business analytics is to identify an insight that will further the organization's goals, and that is also operationally feasible. In business analytics, an insight that satisfies these two criteria is known as an actionable insight. How are control and operational feasibility related to the level at which data is aggregated? Here's how, actionable insights are specific, and obtaining specific insights depends on disaggregated data. Let's look at an example.



Let's say that you want to offer a weekly discount to customers to boost revenue. If you offer a discount that is too big, then the increase in sales volume will not lead to increased revenue, let alone profit. On the other hand, if the discount is too little, then demand will not increase enough to generate additional revenue. Consequently, there's an optimal discount percent that you want to obtain.

To identify the optimal discount size, you would take historical data and quantify the relationship between sales price and sales volume, which is also known as the price elasticity of demand.



Week	Discount	Profit Per Unit	Quantity	Profit
2021-11-07	.05	5.50	1440	8064
2021-11-14	.30	3.00	6250	18750
2021-11-21	.45	1.50	10563	15844

Because you would want to adjust the discount on a weekly basis, you would aggregate the data at the weekly level so that each row represents a week's worth of activity. There would at least be five columns of the aggregated data, date, discount profit per

unit, sales quantity, and profit. So far, so good. You could use this data to find the price elasticity of demand at the weekly level and then adjust the discount each week to obtain that price. This seems reasonable and operationally feasible. Thus, this is an actionable insight. Let's consider two alternative scenarios.

Hour	Discount	Profit Per Unit	Quantity	Profit
2021-11-07 13:00	.05	5.50	14	77.00
2021-11-07 14:00	.30	3.00	62	186.00
2021-11-07 15:00	.45	1.50	105	157.50

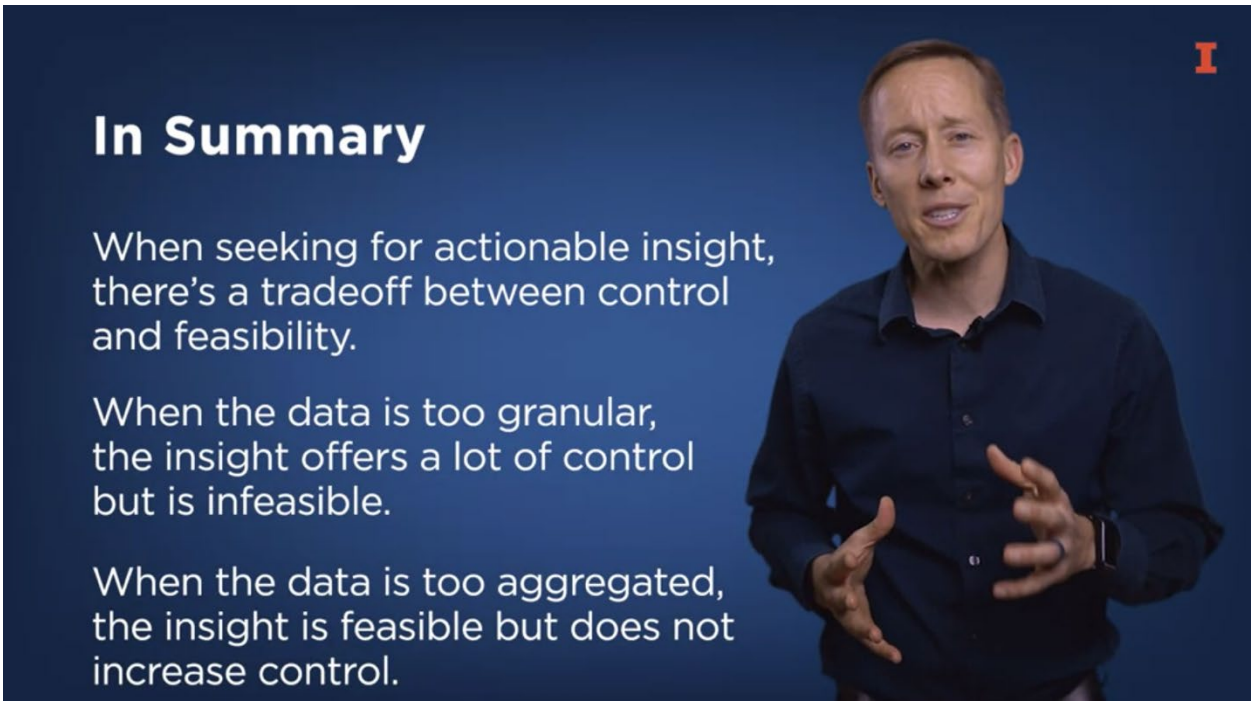
First, identifying the optimal weekly discount may lead you to seek other improvements, such as identifying an optimal discount for each hour of each day. If you had granular point of sale data, you could aggregate the data so that each row represents an hour's worth of data. You could then find the price elasticity of demand at the hourly level and use that to identify an optimal discount for each hour of the day. In theory, this could lead to better profitability. But is it operationally feasible? I think in most cases it would not be, because it would take too much effort to adjust the discount for every hour of every day.

Thus, in this case, the insight is too specific to be actionable.

Year	Discount	Profit Per Unit	Quantity	Profit
2019	.05	5.50	74880	411840
2020	.30	3.00	325000	975000
2021	.45	1.50	549276	823914

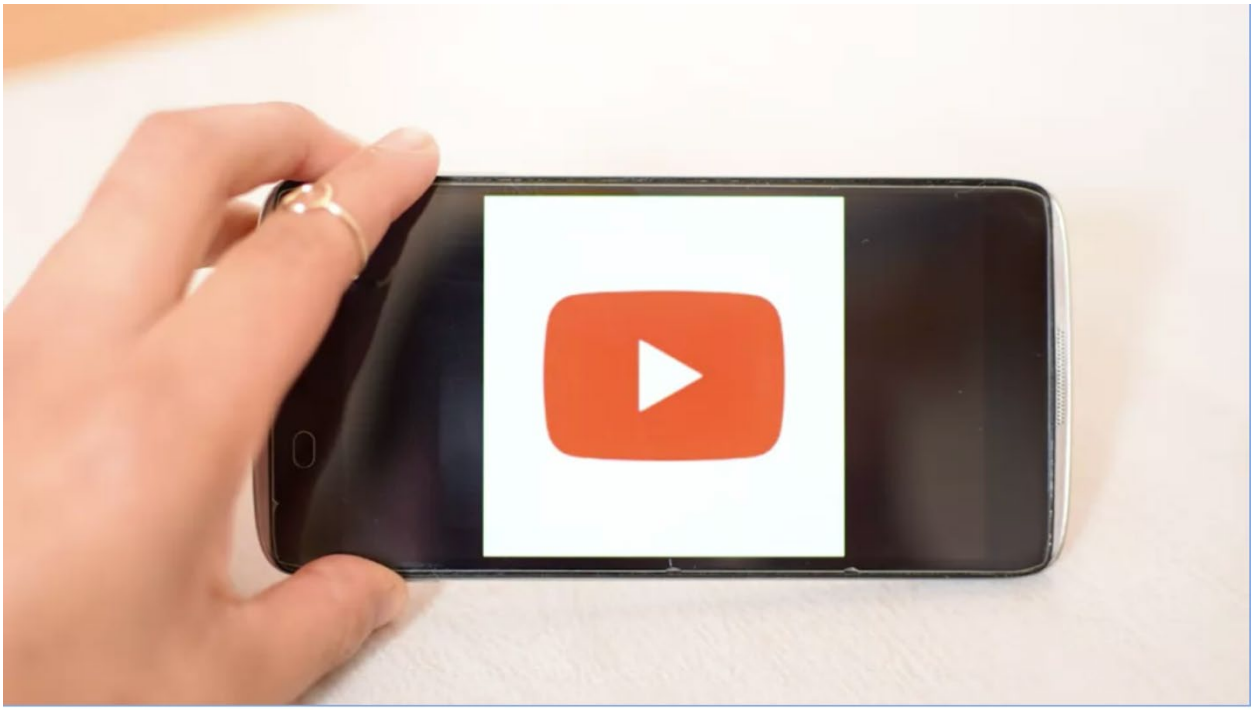
As a second example from the other end of the spectrum, if the raw sales data is only available at the annual level, then each row represents a year's worth of data. If you have enough data, the analysis can only be used to identify the optimal discount on an annual basis.

This is operationally feasible but offers little insight about the ability to control profit. Thus, in this case, the insight does not prescribe enough action. Another problem with data that is aggregated at too high level, is that the details are often lost. What if there is a pandemic that artificially reduces volume for a portion of the year? Or what if there's a local event that causes sales volume to increase regardless of what the price is? Or what If you want to analyze the price elasticity of demand for each department or line item? If data is aggregated at the annual level for all products, there will not be enough detail to refine the estimate for the optimal discount.



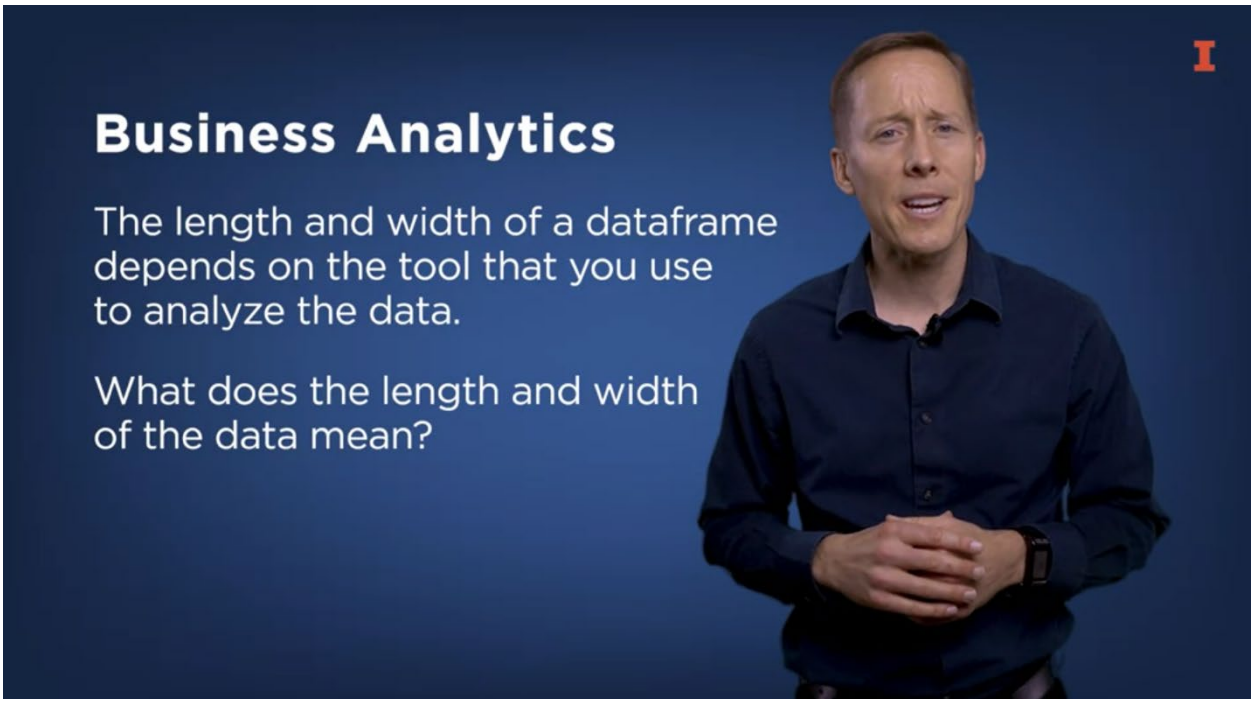
To sum things up, when using business analytics to seek actionable insight, it's worth recognizing the tradeoff between control and operational feasibility. The level at which data is aggregated influences the extent to which you can obtain actionable insight. If the level of the analysis is too granular, then the insight may be too specific to be actionable. Thus, the insight offers a lot of control but is infeasible. On the other hand, if the data is too aggregated, the insight may not direct action enough. Thus, the insight is feasible but does not increase control.

Lesson 4-1.5 Dataframe Shape: Wide Versus Long

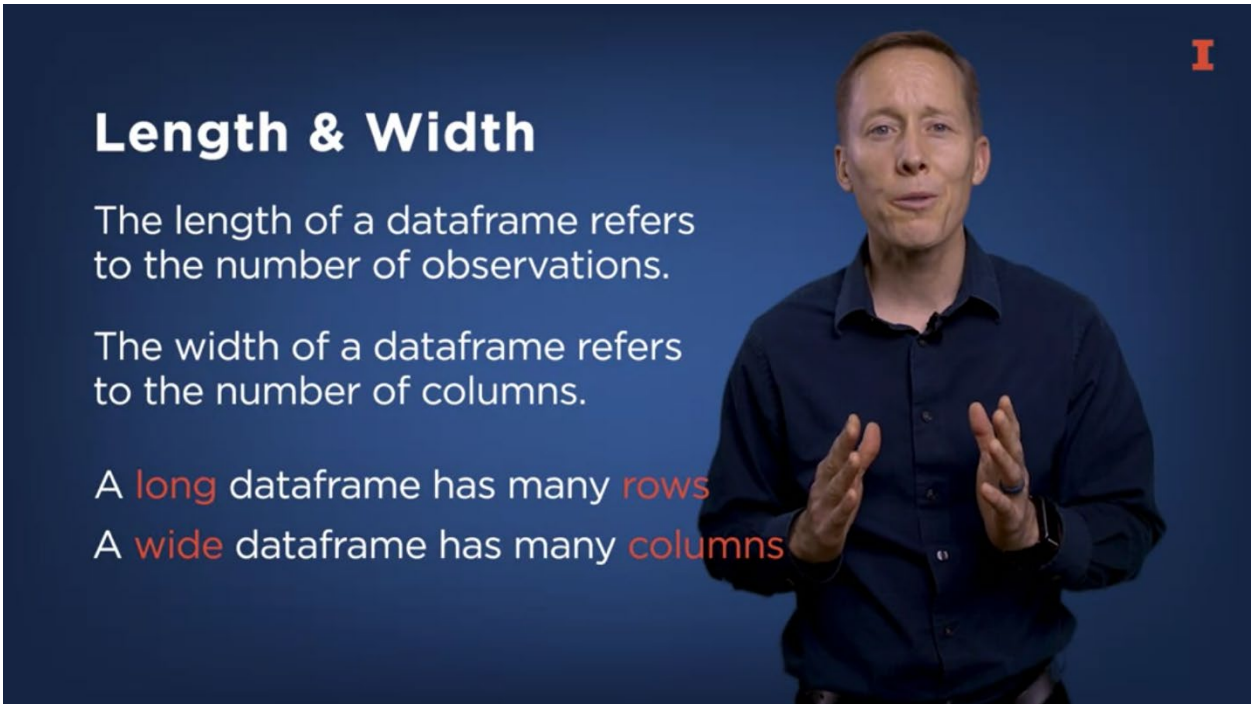


In this lesson, we'll discuss the length and width of the data. Let's liken this topic to the way videos are recorded using smartphones. Based on my observation, it used to be the case that most people preferred holding the smart phone horizontally when taking videos, because they planned on viewing the video on YouTube with a TV or computer monitor, which is wider than it is long.

More recently, people often prefer to hold their smartphone vertically when recording videos, because they plan on posting the video to social media platforms like Instagram and TikTok that are often viewed using smart phones, and smartphones are usually held so that the screen is longer than it is wide. Consequently, the aspect ratio that people choose to use when they record videos is often influenced by the viewing platform for which the video is intended.

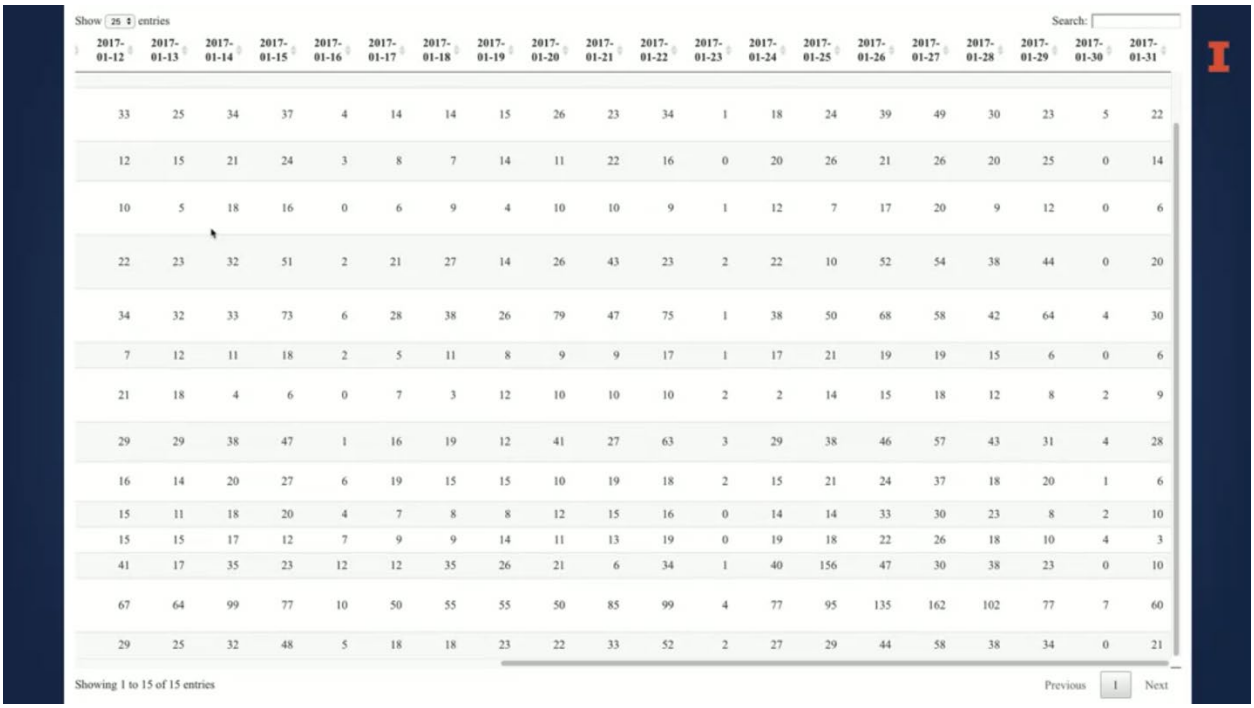


In a similar manner, when it comes to business analytics, the length and width of the data depends on the tool that you plan to use to analyze it. What does the length and width of the data mean?



Because every row of a tidy dataframe represents an observation, the length of a dataframe refers to the number of observations it has. Because each column of a tidy dataframe represents a unique characteristic of the observation, the width refers to the

number of columns. Thus, when we talk about a long dataframe, we are referring to a dataframe that has many rows. In contrast, when we talk about a wide dataframe, we're referring to a dataframe that has many columns. Business analytic tools typically make it pretty easy to pivot the data from a wide format to a long format and vice versa.



Let's look at an example in which each row of data represents a line item, one column lists the name of the line item, and then each of the other 31 columns represent the sales quantity for that line item for a specific day of the month in January. If we have 15 line items and 31 days of sales, then this dataframe is relatively wide and not very long.

457	Yogurt	2017-01-23	2
458	Yogurt	2017-01-24	27
459	Yogurt	2017-01-25	29
460	Yogurt	2017-01-26	44
461	Yogurt	2017-01-27	58
462	Yogurt	2017-01-28	38
463	Yogurt	2017-01-29	34
464	Yogurt	2017-01-30	0
465	Yogurt	2017-01-31	21

Showing 401 to 465 of 465 entries

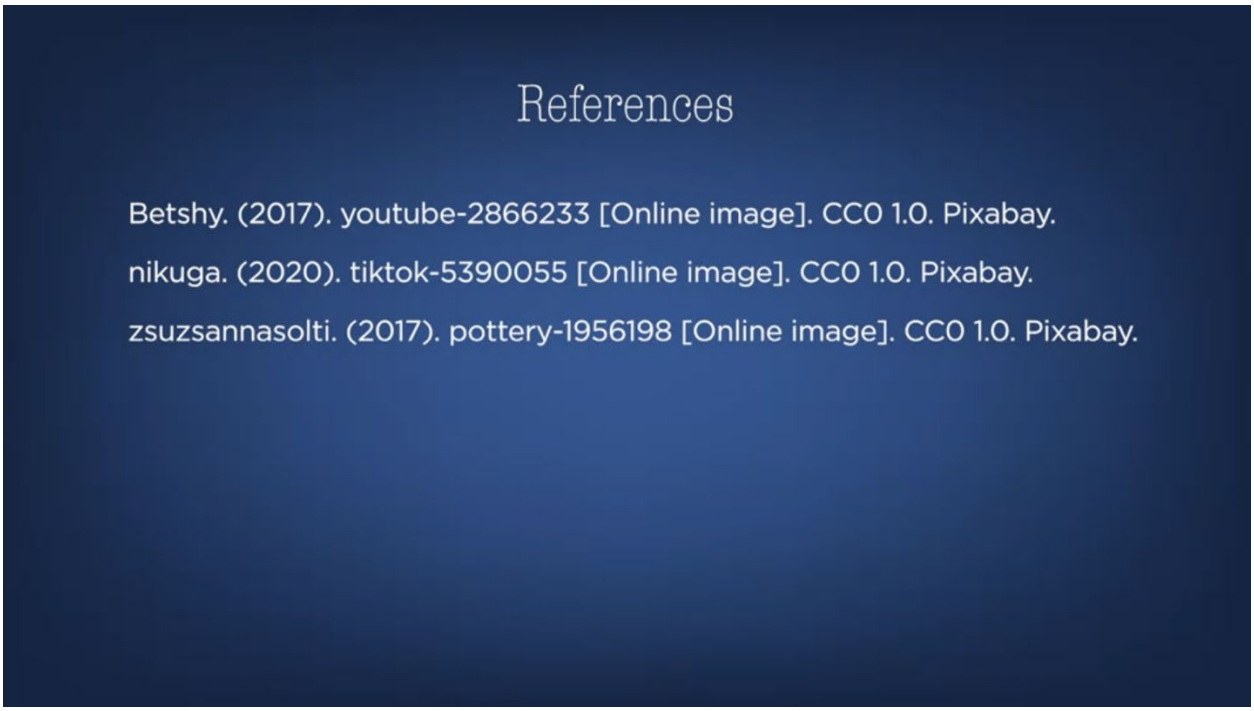
Previous 1 2 3 4 5 Next

We can pivot that dataframe to create one in which each row represents a different date for each line item and the only additional column represents the total sales for that line item on that day. If we have 15 line items and 31 days of sales for the month of January, then this dataframe can have up to 465 rows and only three columns. This version of the dataframe is relatively long.

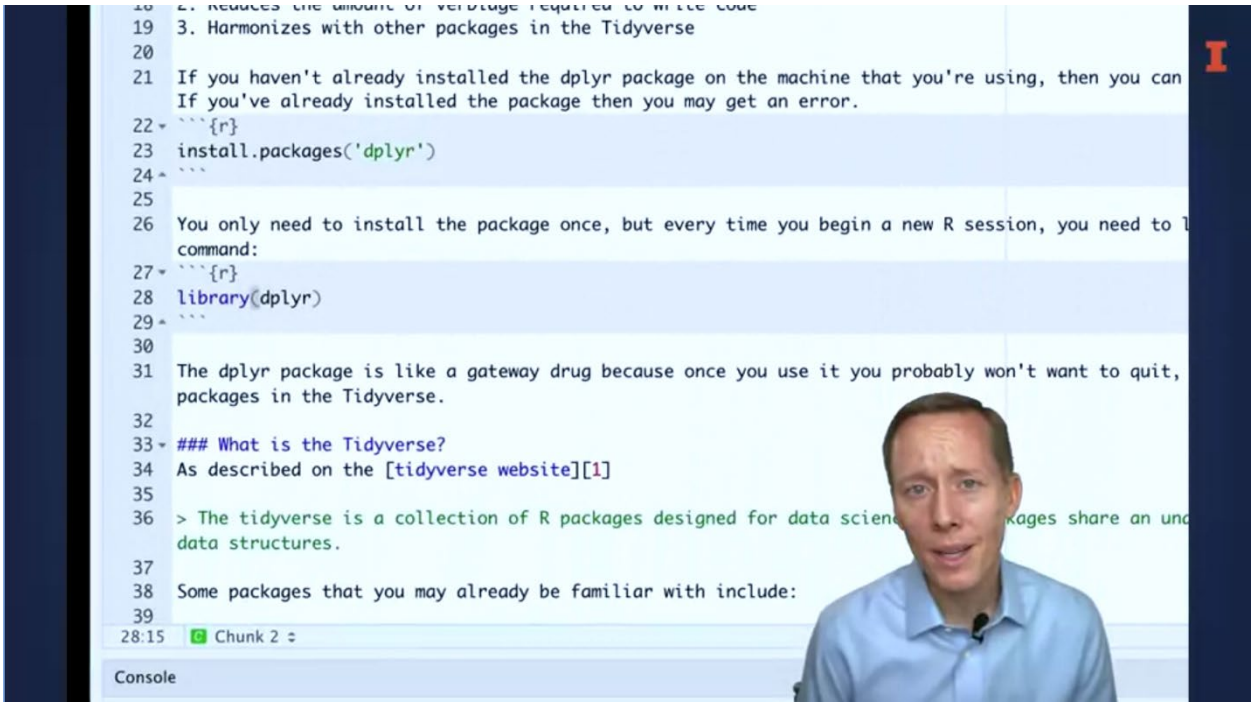
How you intend to analyze the data decides its length and width. Typically, visualization

software and dashboards rely on long data formats, while algorithms and tables intended for human consumption, often rely on wider data formats.

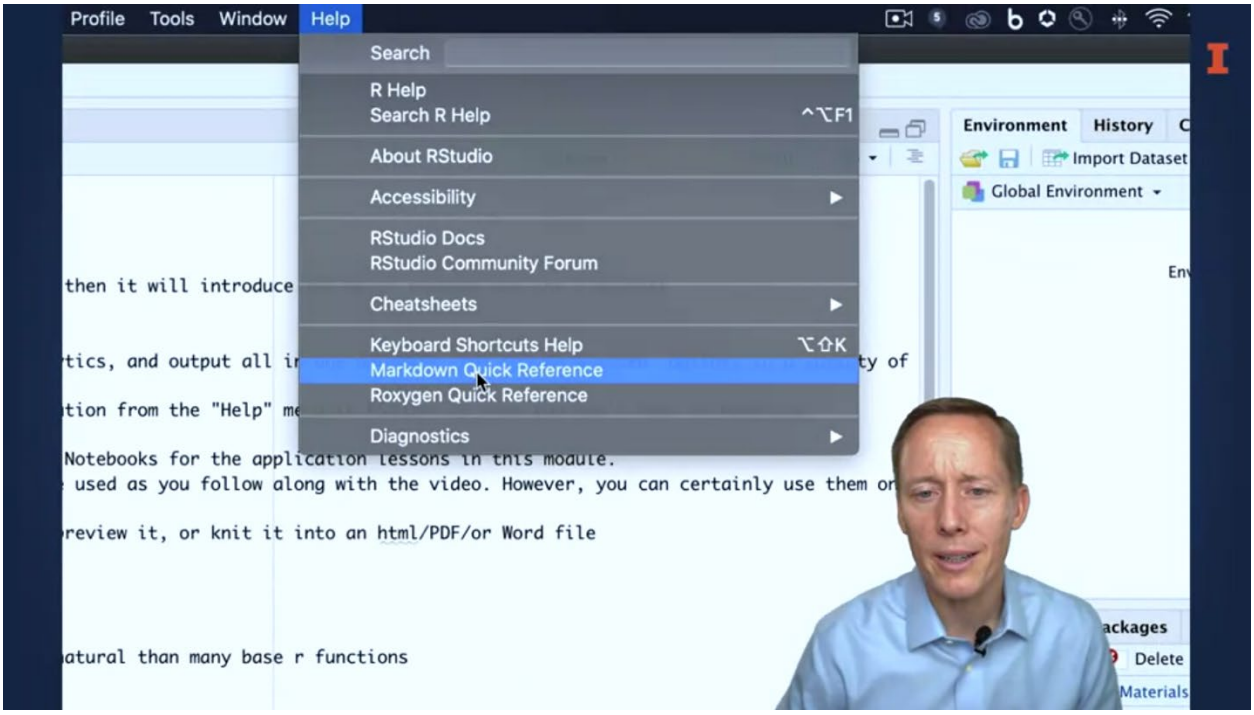
In conclusion, as you assemble data for calculations, the first step is to tidy up the data by filling in missing values and making sure that the format of the data in every column is consistent. Once you have a tidy dataframe, you should pre-process the data so that it's in a format that's suitable for the question that you have framed. One element of data pre-processing is making sure that the data is aggregated at the level that is consistent with the question. Another element of data pre-processing is to pivot the data to the length and width that will best fit the analytic tools that you plan on using.


References
Betshy. (2017). youtube-2866233 [Online image]. CC0 1.0. Pixabay.
nikuga. (2020). tiktok-5390055 [Online image]. CC0 1.0. Pixabay.
zsuzsannasolti. (2017). pottery-1956198 [Online image]. CC0 1.0. Pixabay.

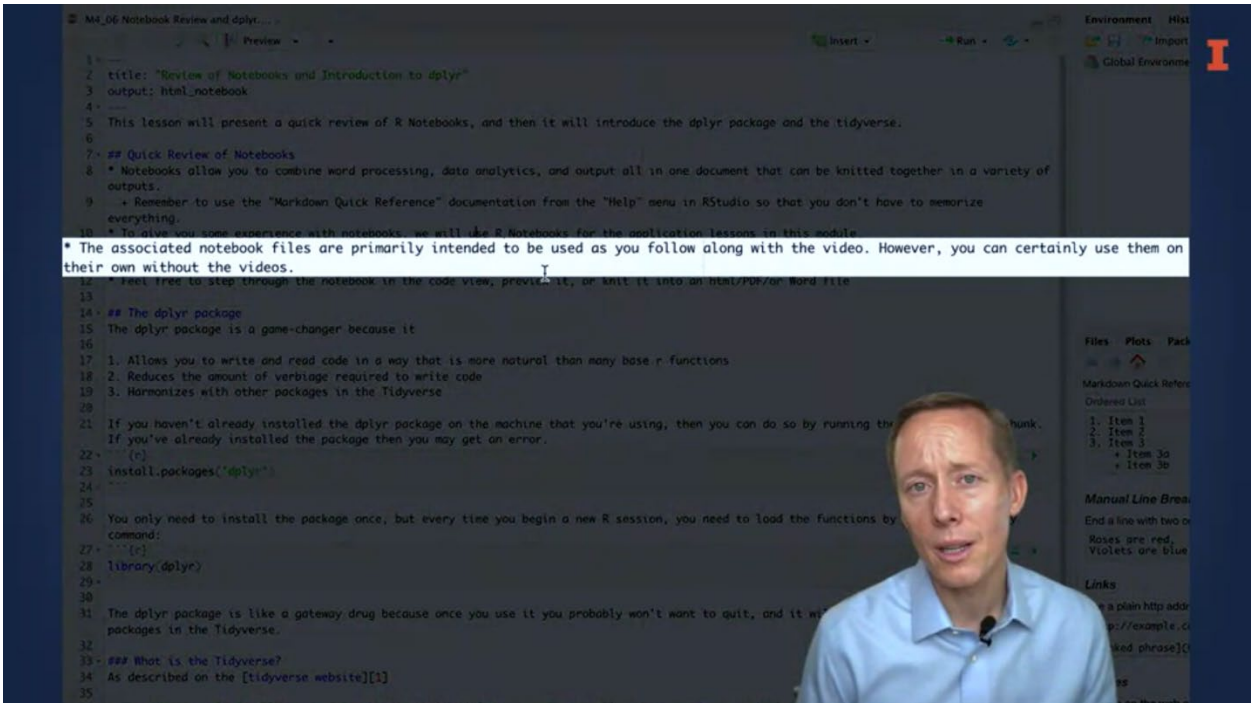
Lesson 4-1.6 Review of Notebooks and Introduction to dplyr



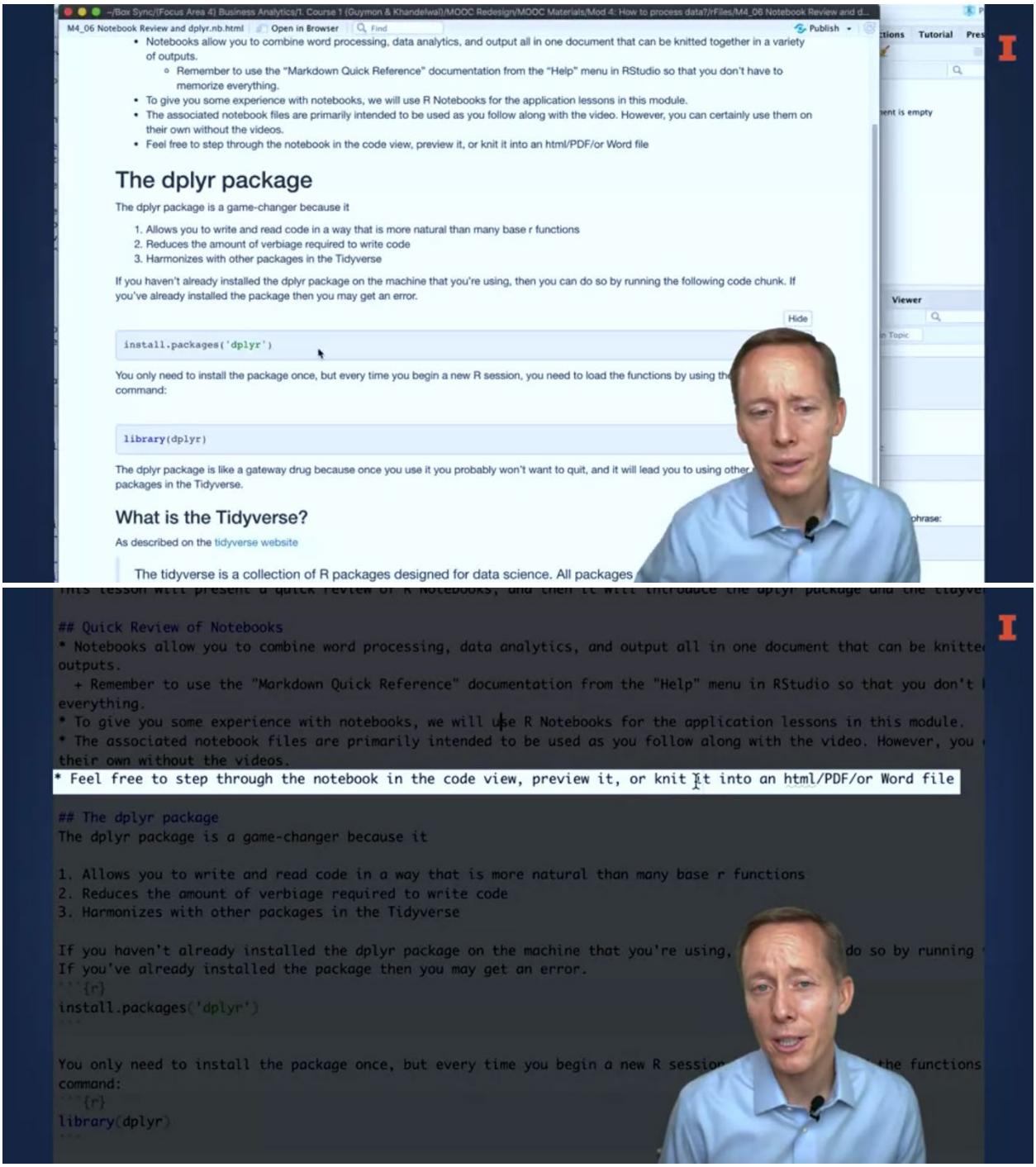
In this lesson, I want to quickly review our notebooks and also introduce you to the dplyr package as well as the tidyverse. I want to review our notebooks with you because up to this point the lessons have used our script files. Now in this lesson and all the other lessons in this module, we're going to use our notebooks. As a reminder, notebooks allow you to combine word-processing, data analytic, and visualization capabilities all within the same file. You do not have to memorize all of the formatting options.



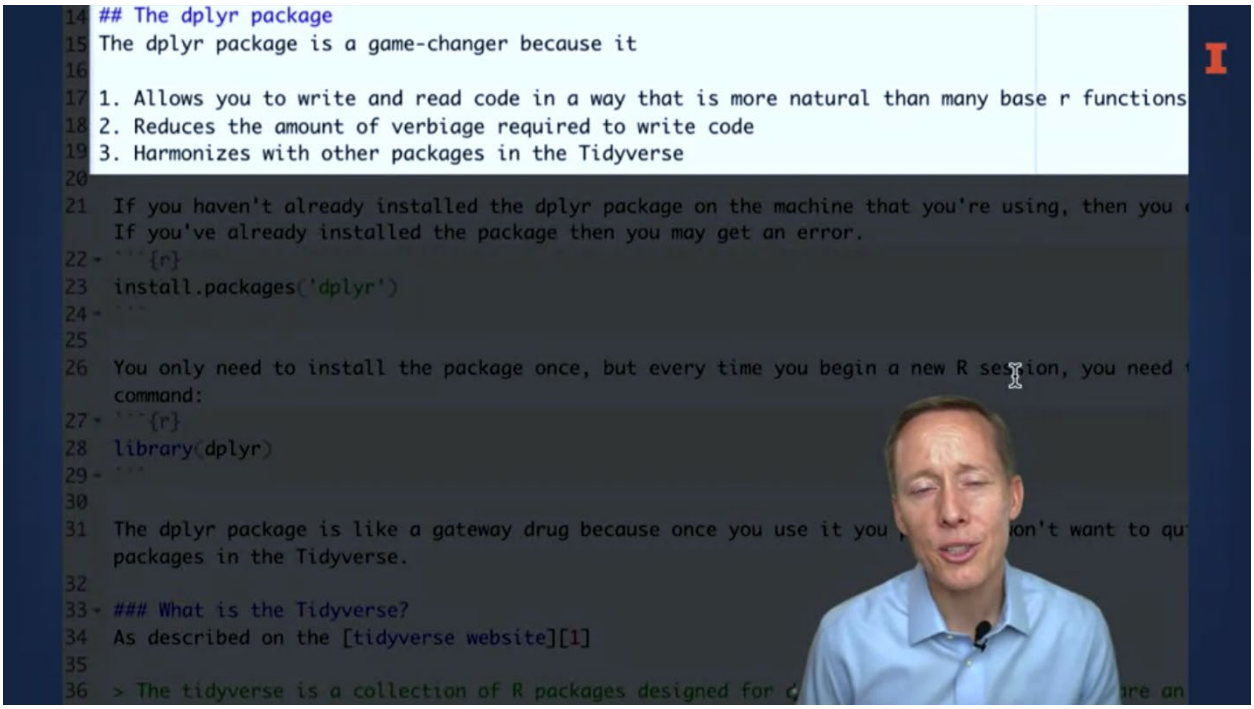
Remember to use the Markdown quick reference guide, which will present to you in the Help pane how you can format your text.



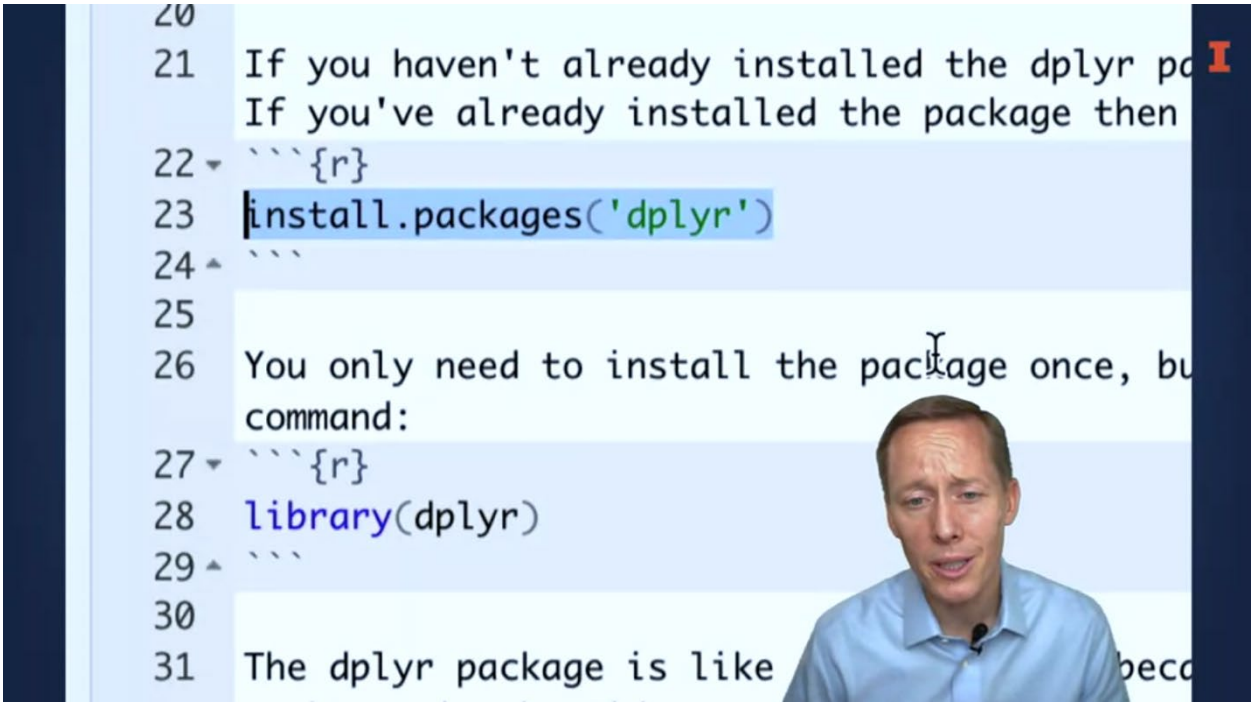
Now these files are intended primarily to be used as you watch the videos. However, they stand on their own. If you want to just read through the Notebook files, you can do that as well.



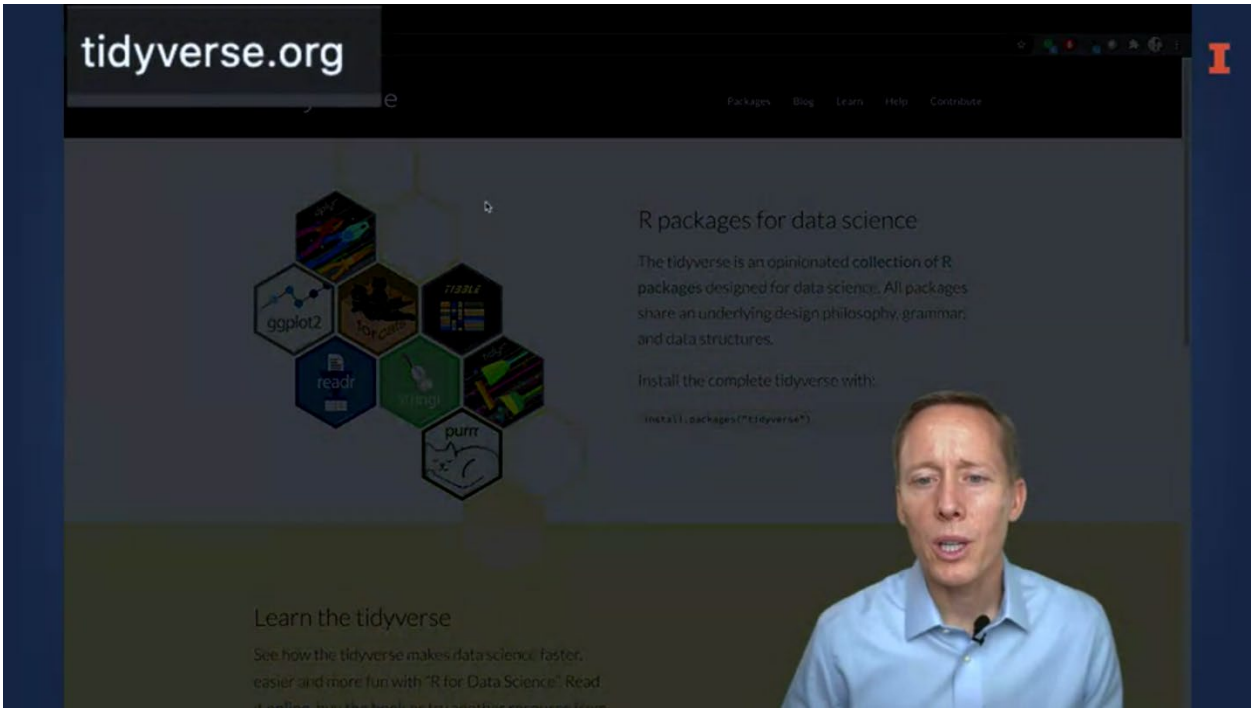
Feel free to either walk through the Notebook files in the Code View or preview the files so that you can see what the formatted text will look like, as well as the code chunks or knit these Notebook files together into an HTML file, a PDF file or a Word file.

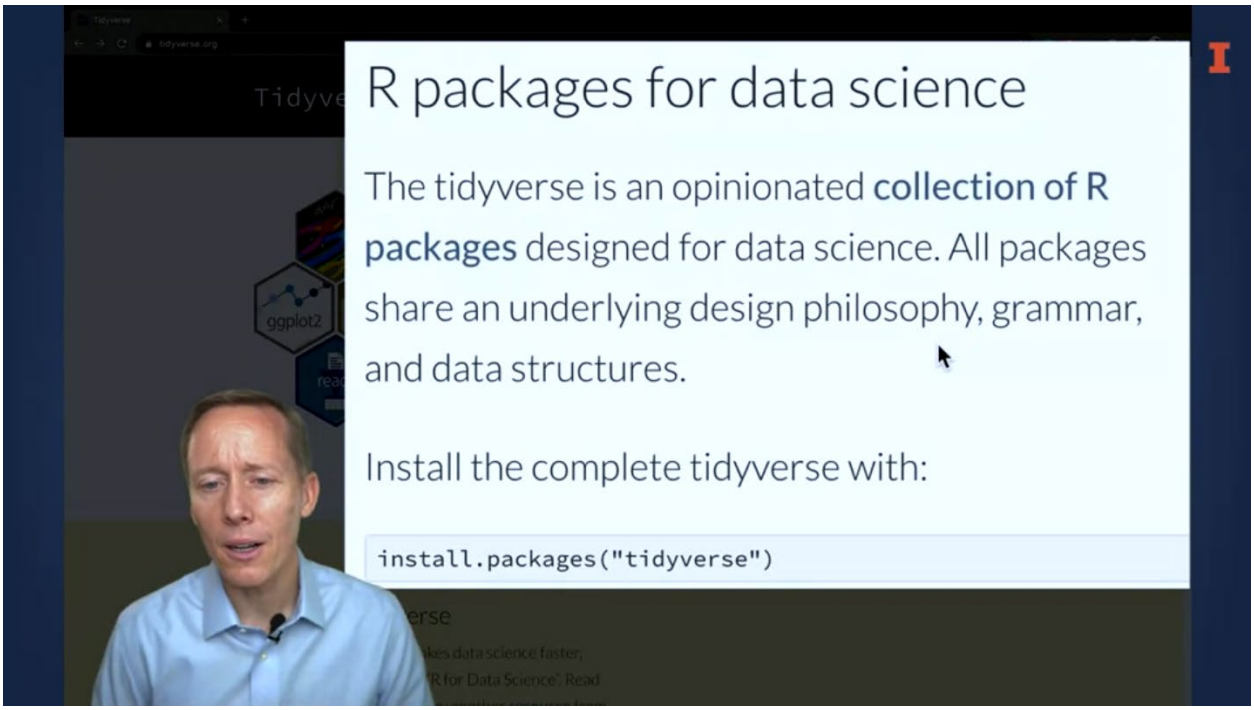


Let's talk about the dplyr package now. The dplyr package is a game-changer because it allows you to read and write code in a way that is more natural than many of the base r functions. A second reason why it's awesome is because you can perform many of the same tasks that you would perform with base r functions, but with much less verbiage. Finally, the dplyr package harmonizes with other packages in the tidyverse, and it makes all of them together very powerful.

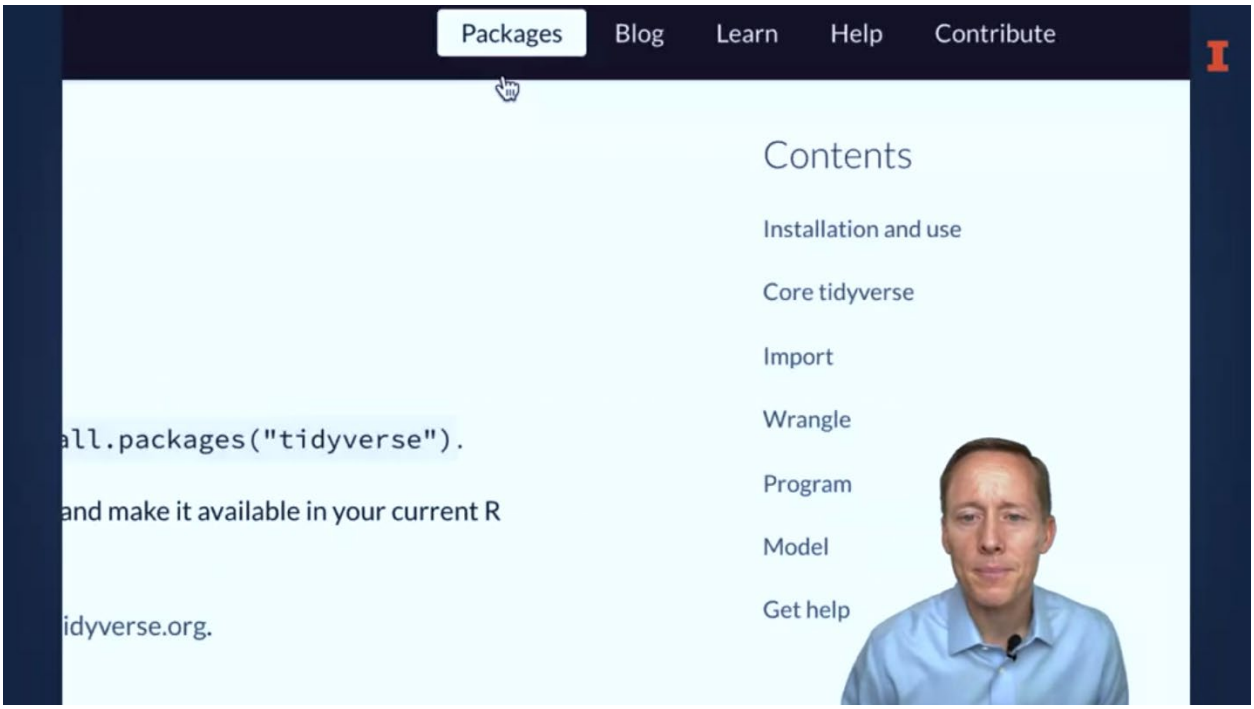


You only need to install the dplyr package once. So if you haven't already done so, make sure and then run the install.packages command for dplyr. Once you've installed it, you don't need to run it again and if you have it in a code chunk and it's already installed, you may get an error. Now in order to use the functions in the dplyr package, as with all packages, you need to make sure and use the library function to call those functions to memory. Let's talk about the tidyverse. What is the tidyverse?

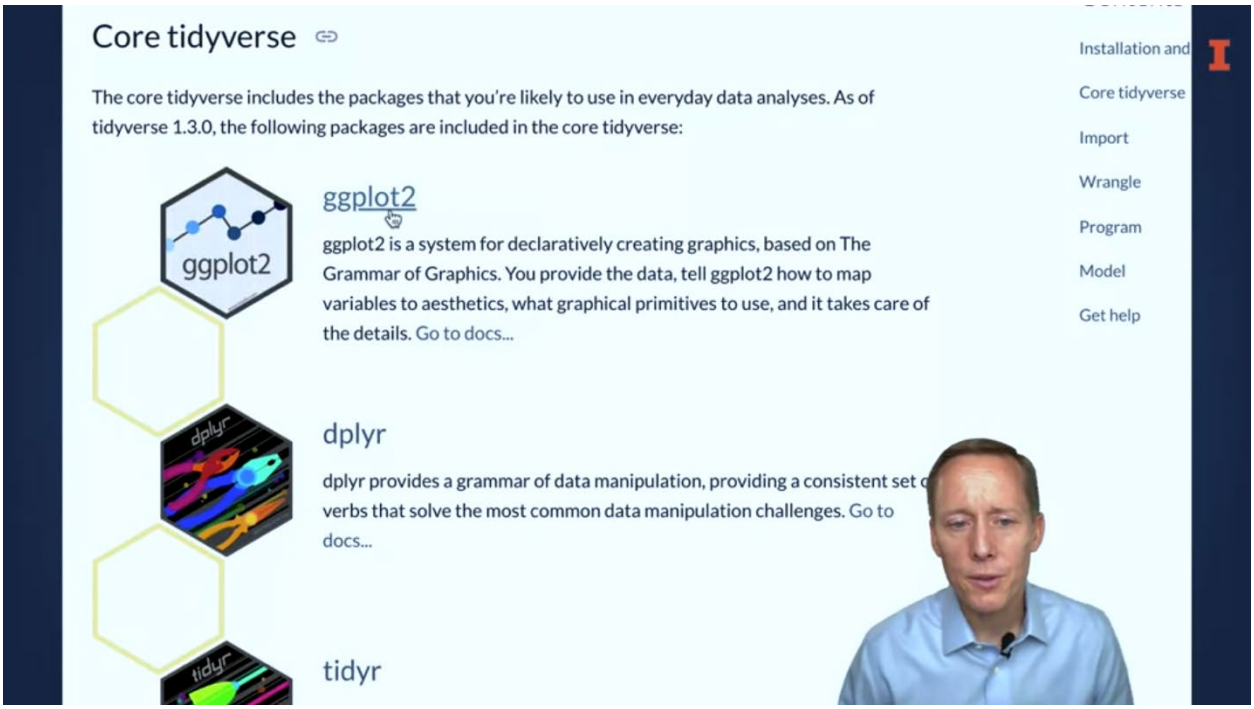




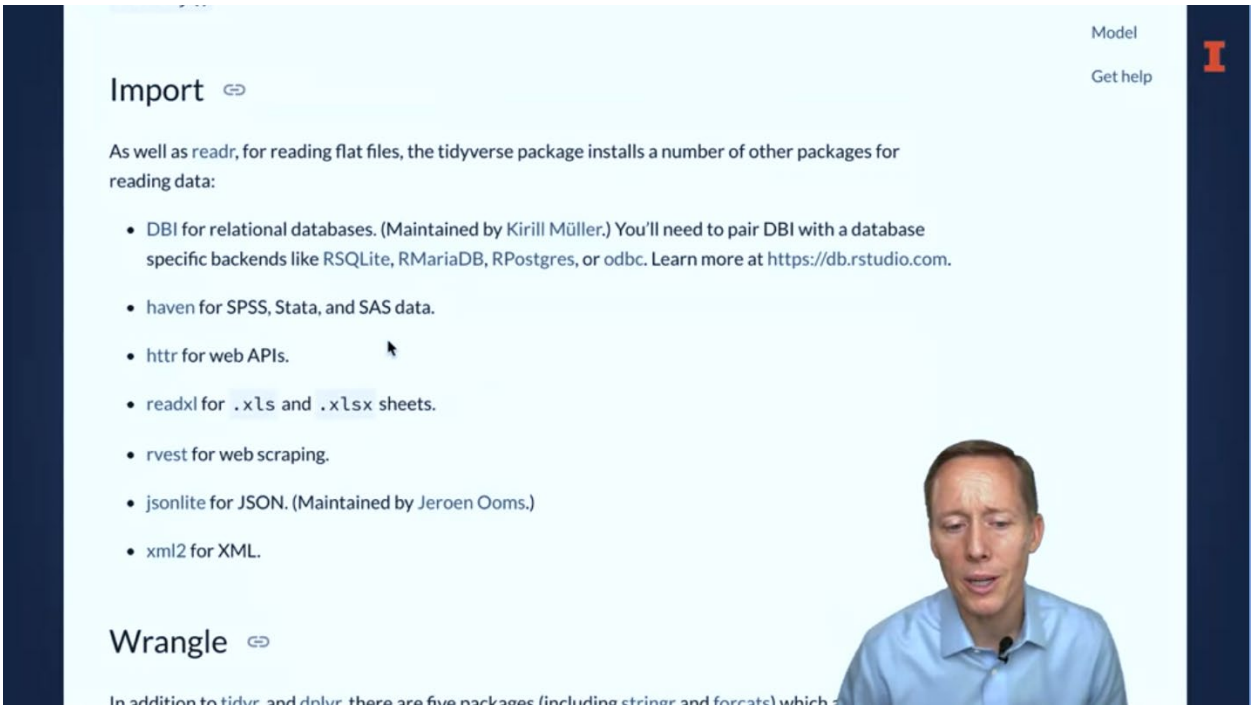
If we go to the tidyverse website, it says the tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. You can install the complete tidyverse with this install.packages function right here.



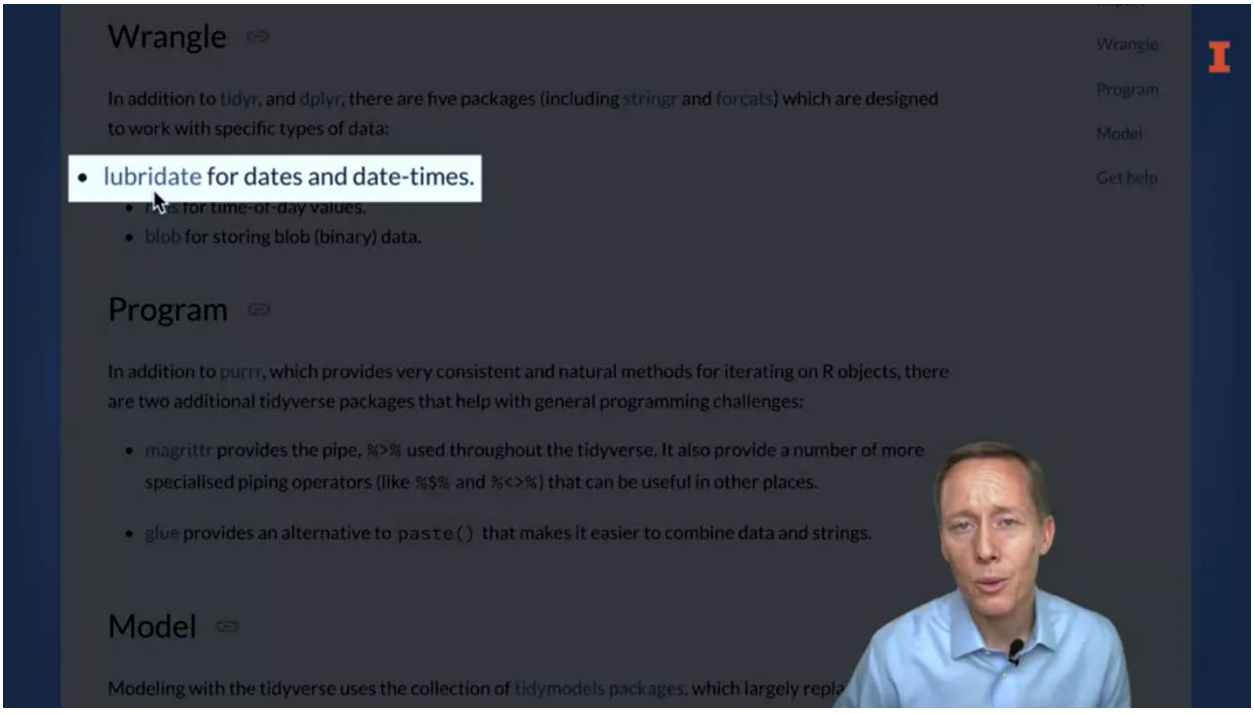
Now, let's go ahead and explore what packages are in the tidyverse.



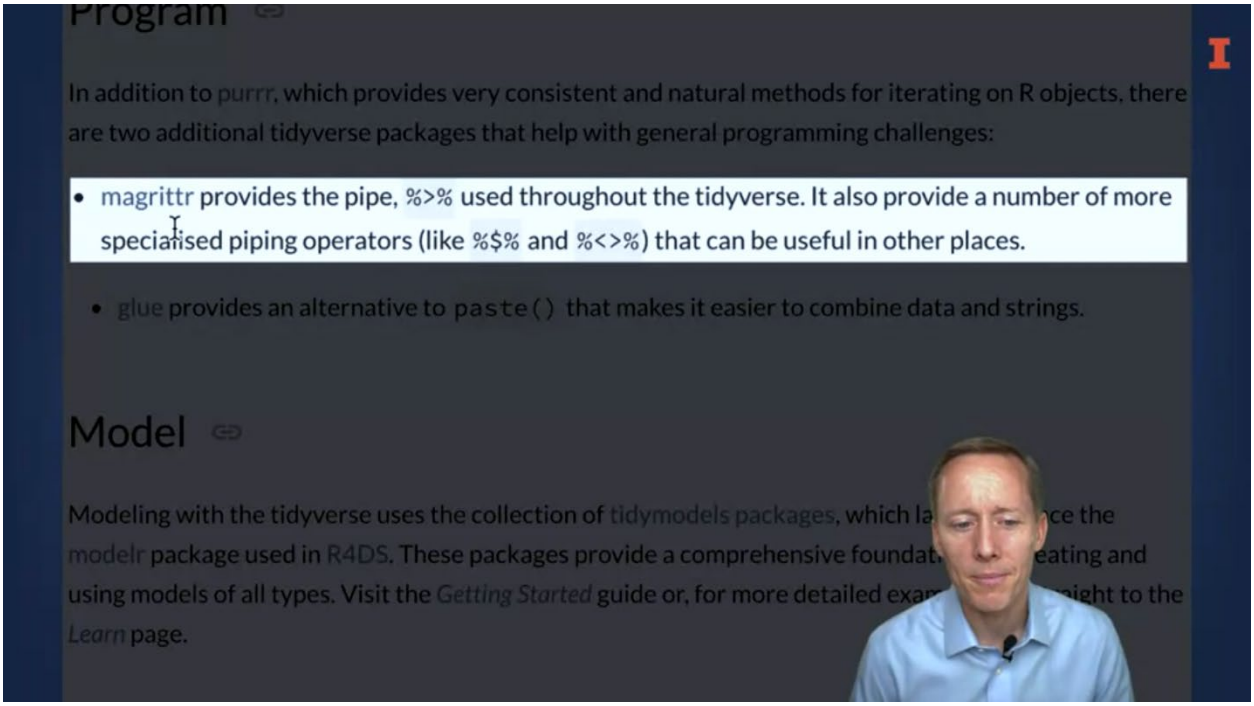
In the core tidyverse, we have ggplot2. That's for creating graphics, dplyr, which we're going to be using, tidyr, we'll use this package in some other lessons in this module, readr, purr, tibble, stringr, which you may have seen before, and forcats which you also may have seen before.



Now in addition to those packages, there are some other packages that are required to use these packages in the tidyverse and those are right here. There are also some packages that are not included in the tidyverse, but the tidyverse packages are designed to work with them.



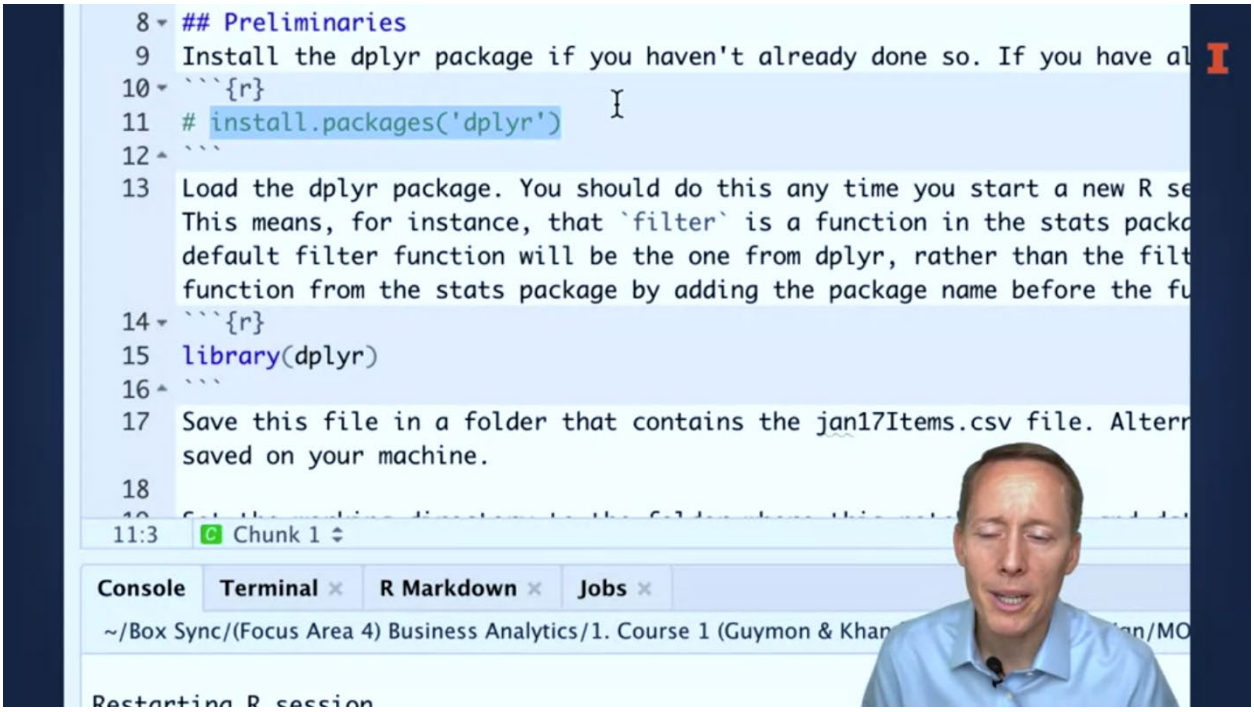
One of those packages is lubridate, which you have probably already used before.



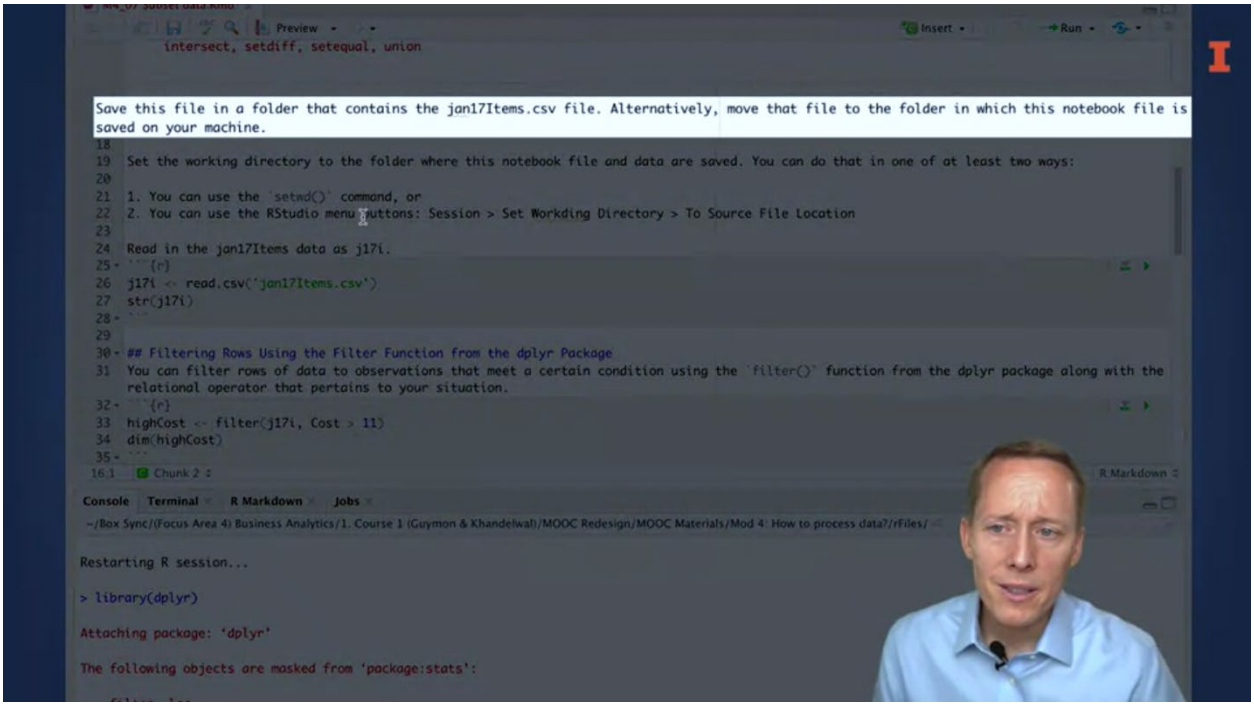
In this module, we will also use the magrittr package to help us write code that is easier to read. Now I encourage you to explore other packages and other aspects of the tidyverse, but that should be a good enough start for us at this point.

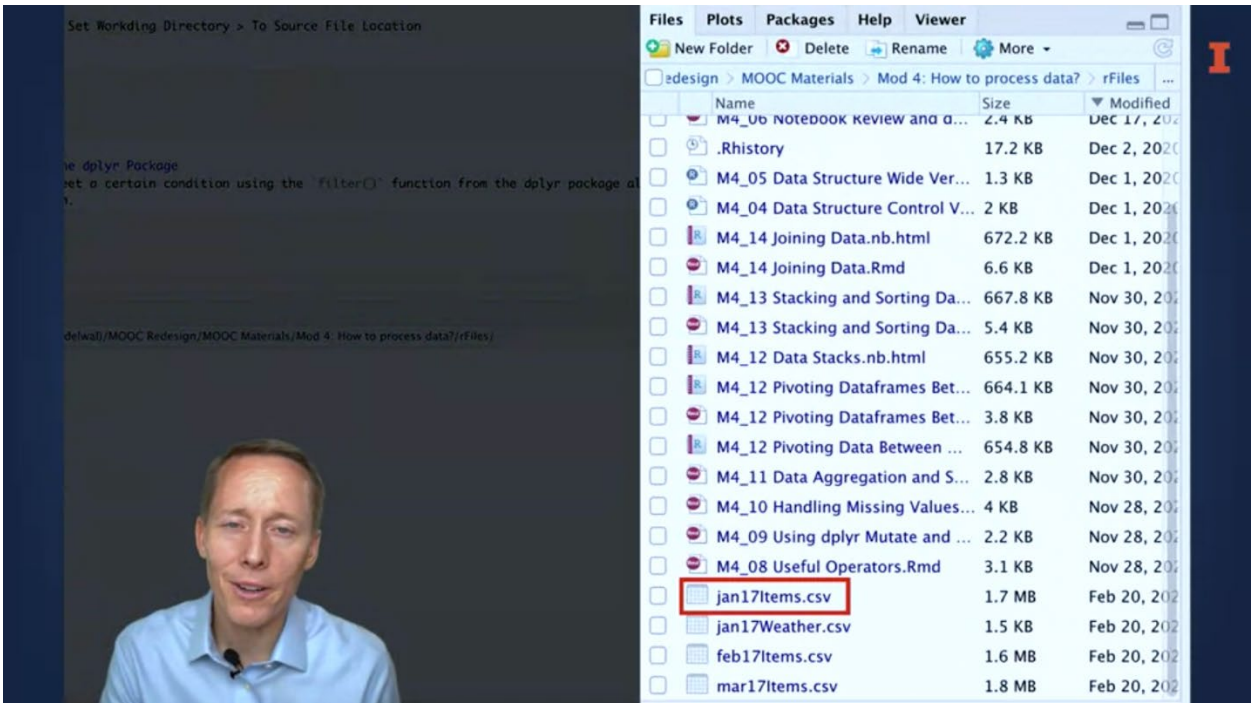
Lesson 4-1.7 Subset Data Using dplyr's Select and Filter Functions

In this lesson I want to show you how to use tidyverse grammar to reduce the size of a data frame. We'll talk about how to reduce the number of rows of a data frame as well as the number of columns of a data frame. You don't have to use the dplyr package to do this, but using the dplyr package makes it much easier.

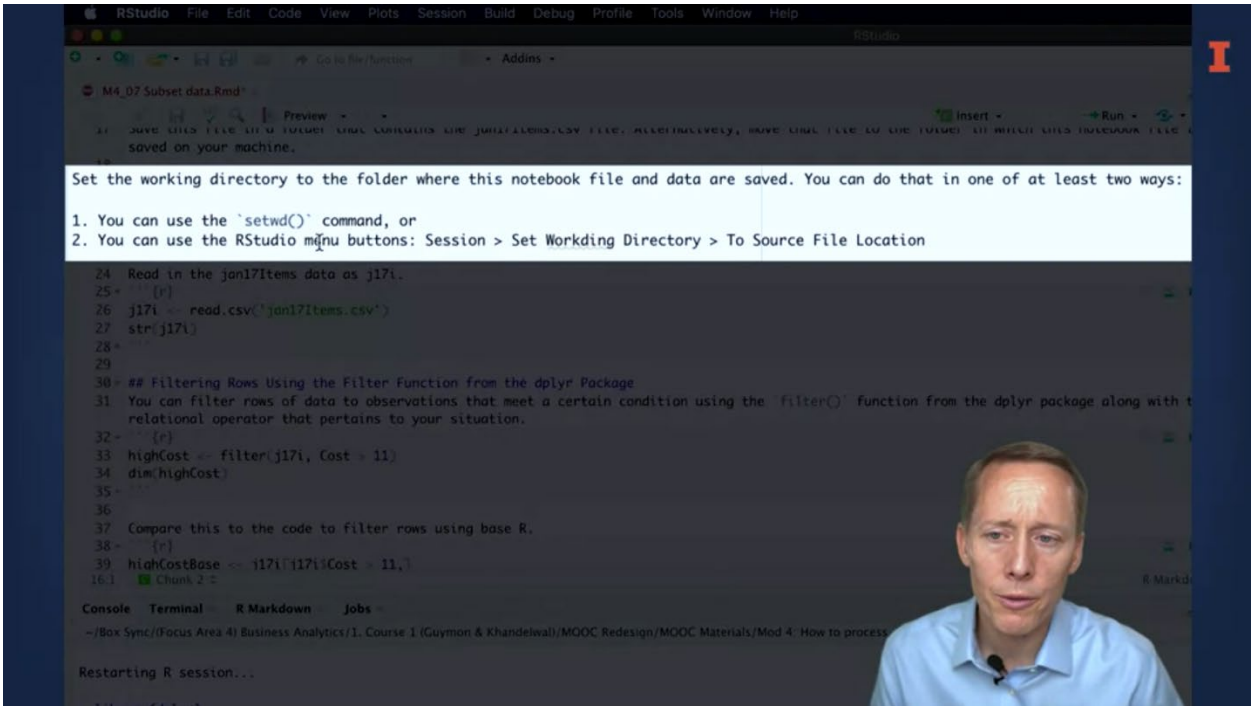


First, if you haven't already done so, please make sure to install the dplyr package. Next, make sure and load the dplyr package, the functions in the dplyr package by using the library command.

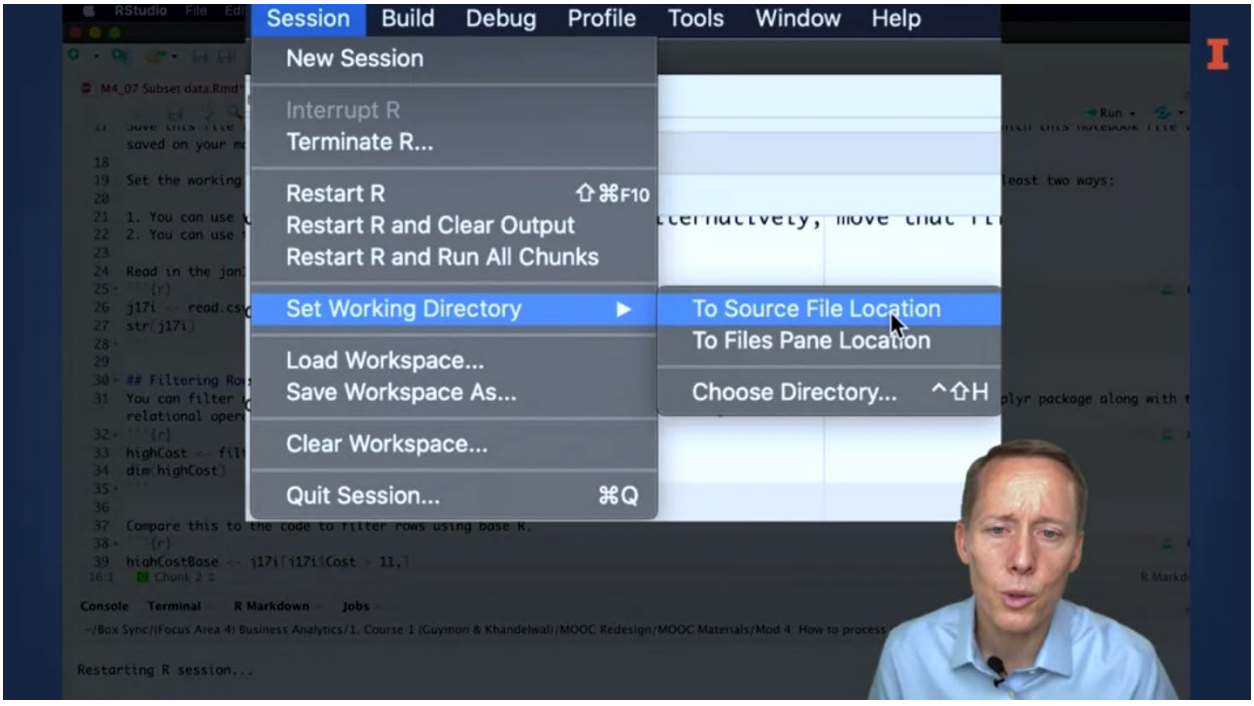




Now, the next thing you want to do is to make sure and save this markdown file, if you're following along in the same directory or folder where you have the jan17items.csv file or remove that jan17 items.csv file into the same folder where this file is.

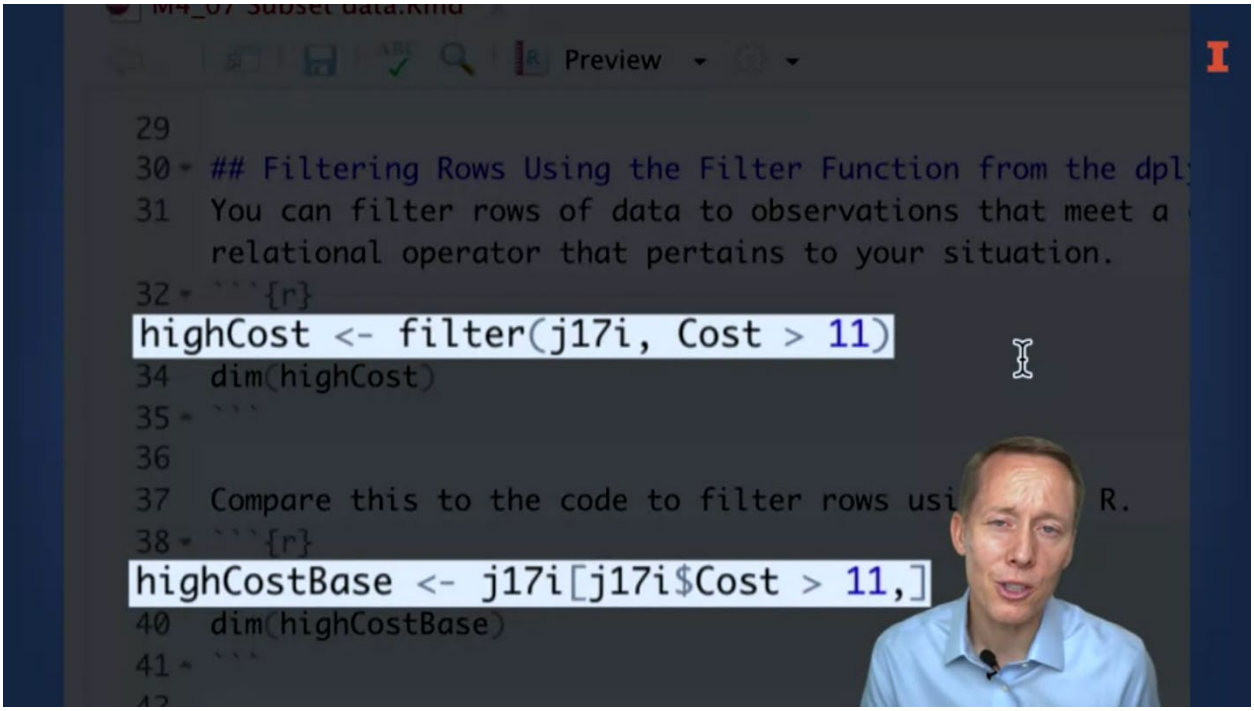


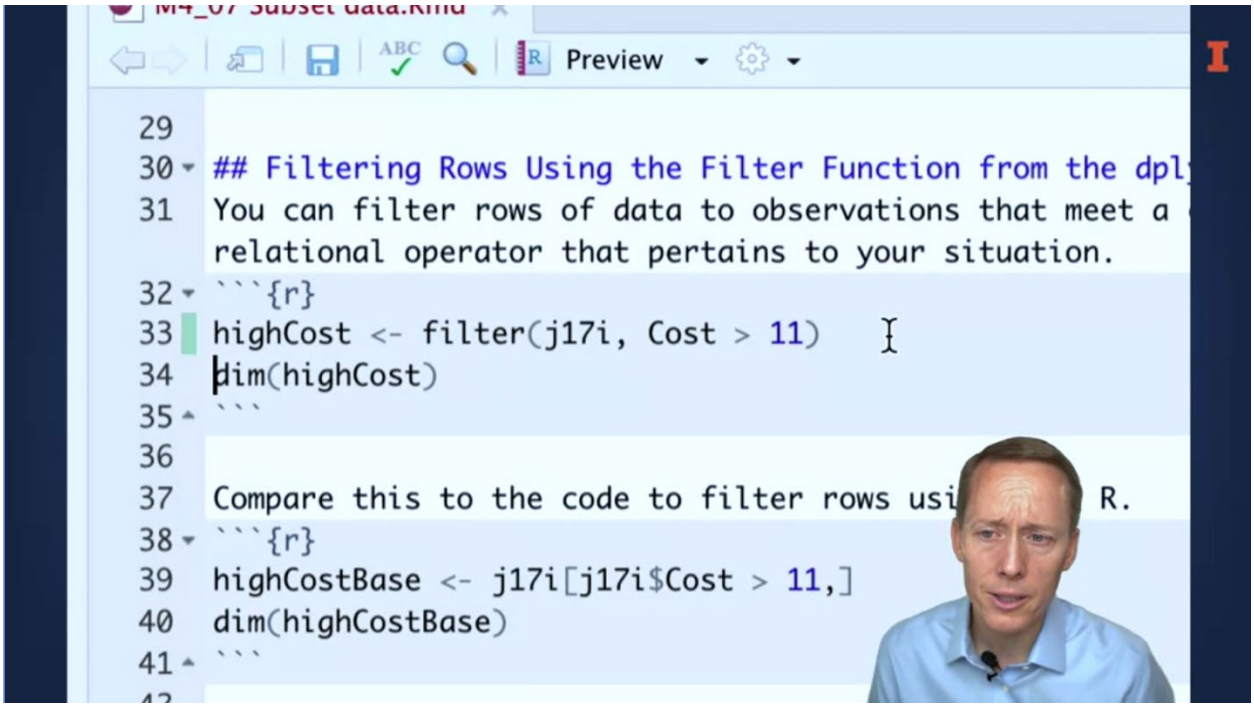
Once you've done that, you can set the working directory to the location of this file. You can do that in one of two ways, you could use a setwd command.



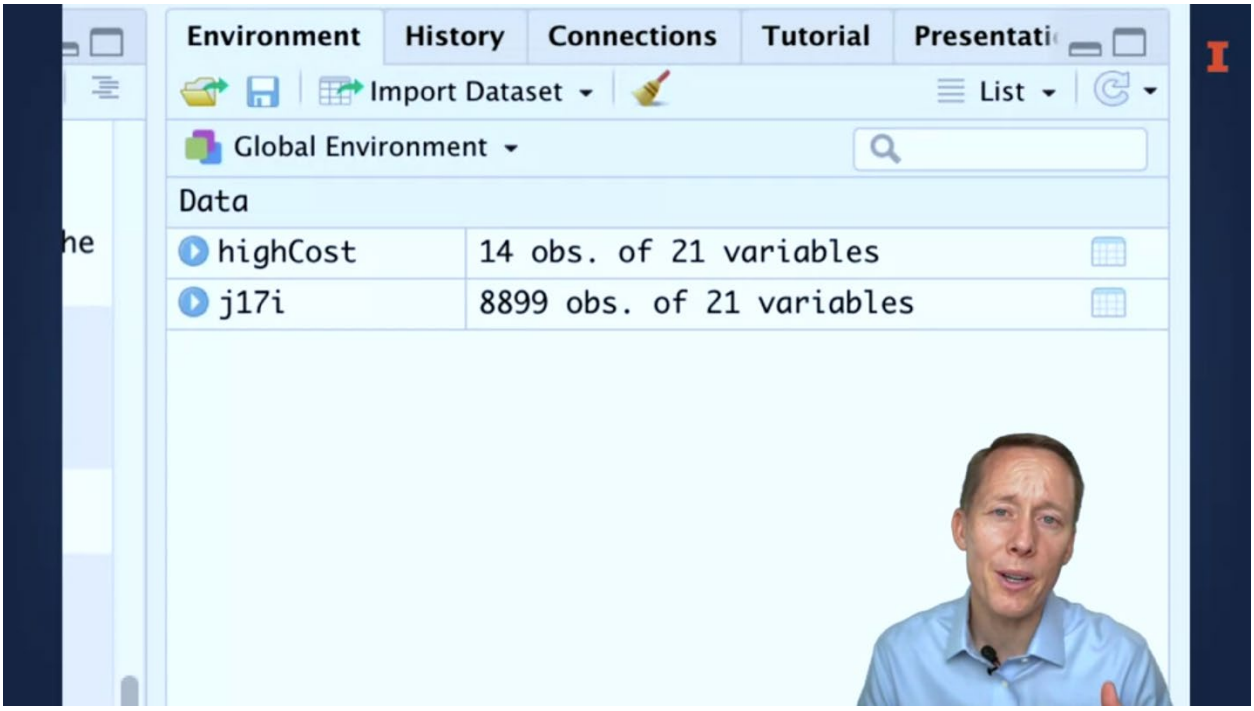
I find it to be easiest to just go up to the Session and go down to Set Working Directory to Source File Location. So now I'll go ahead and read in the jan17items.csv file and store it as a j17i object.

Now let's just evaluate the structure of this data frame using the str command. We can see that we've got 8899 observations and 21 variables. Okay, very good. And there is a preview of the values in each of the columns. So what I want to show you now is how you can reduce the number of rows based on meeting a certain criteria.

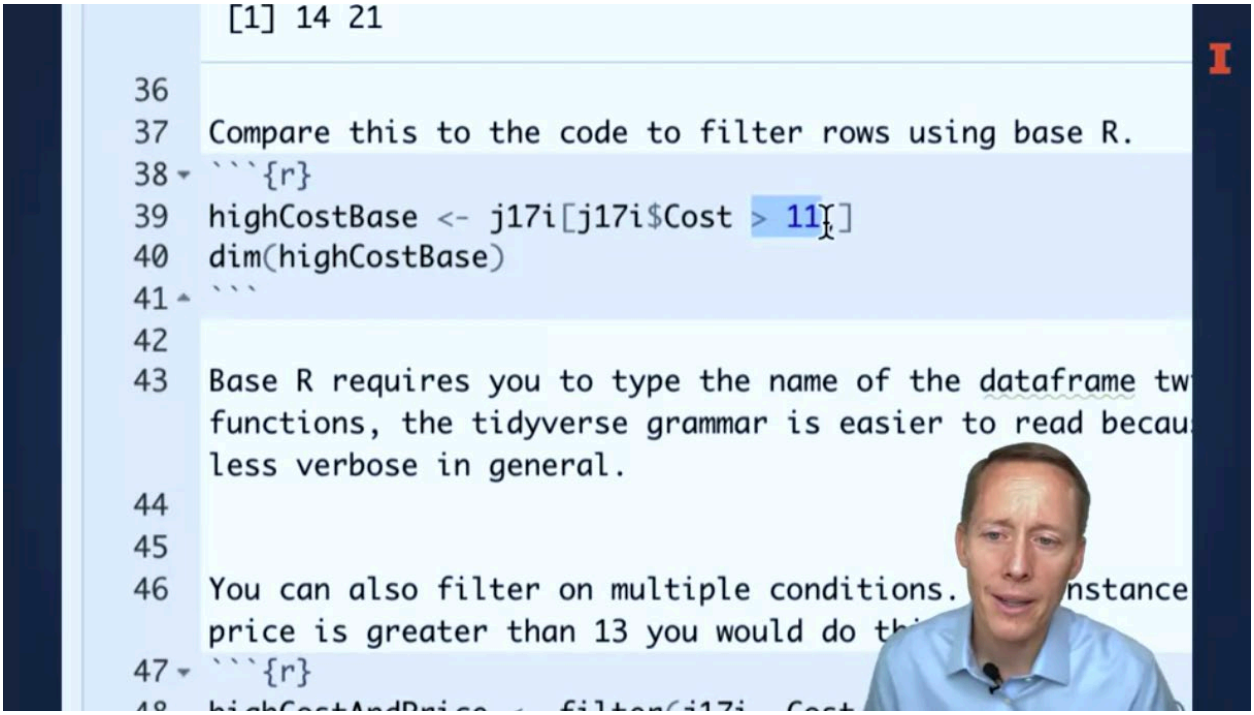


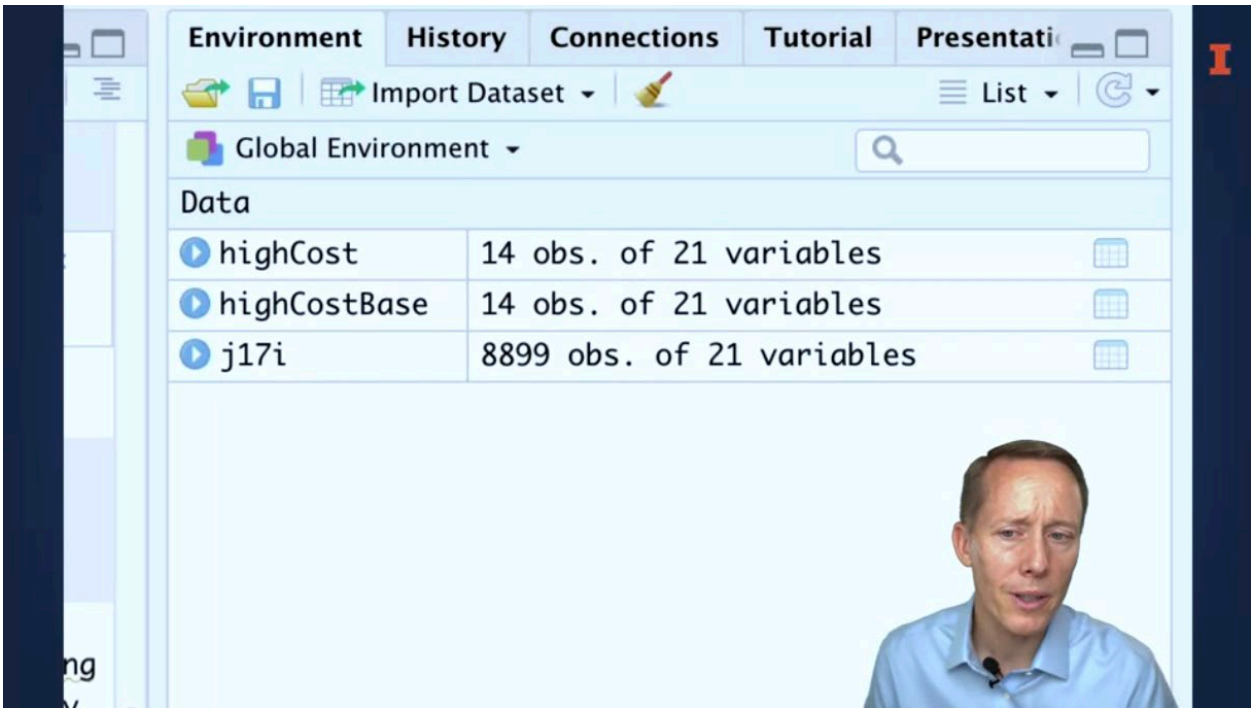


And I'll illustrate how to do this using both the filter function from the dplyr package, as well as using base R commands. So let's say for instance, that we want to filter these observations down to just the observations where the cost is greater than 11. We can do that using this filter function. Then the first argument is our data frame object, and then we indicate what the criteria is for staying in this data frame. And in this case, I'm going to say the value in the cost column has to be greater than 11. And I'll assign that to new data frame object called high cost.



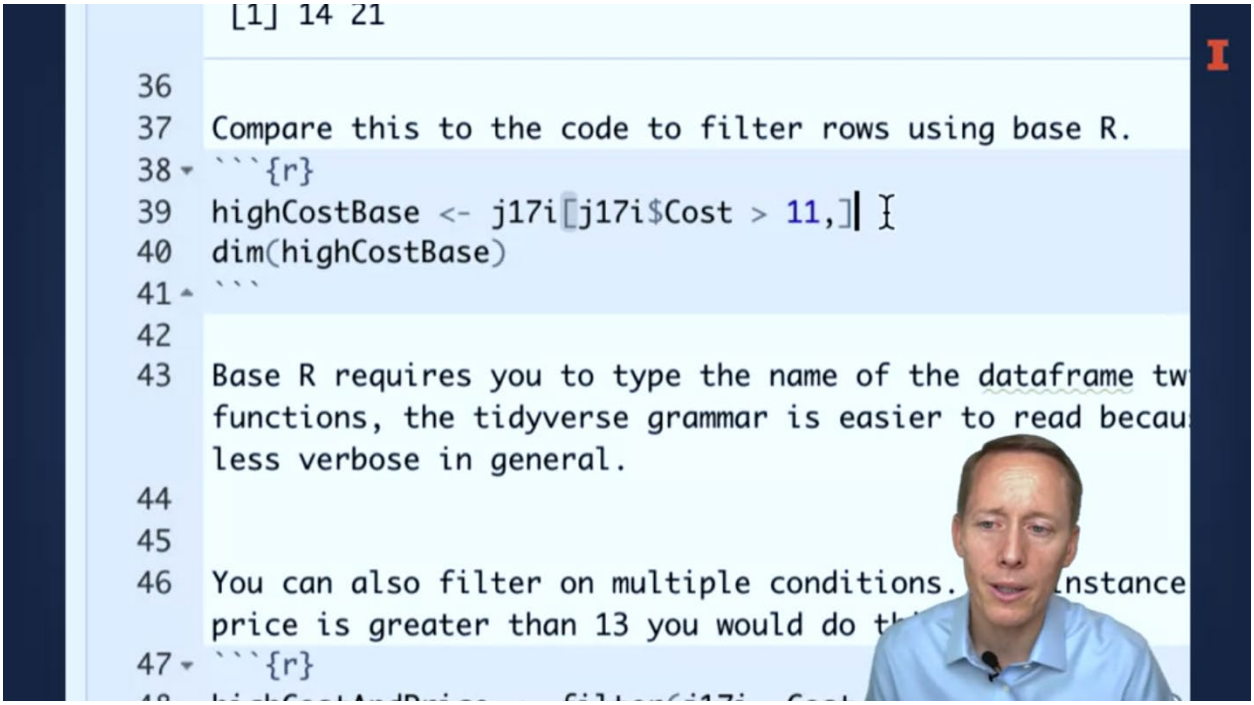
Okay, very good. Now you can look over in the environment and see the high cost exists. There's only 14 observations with the 21 variable, so we've reduced the number of rows.



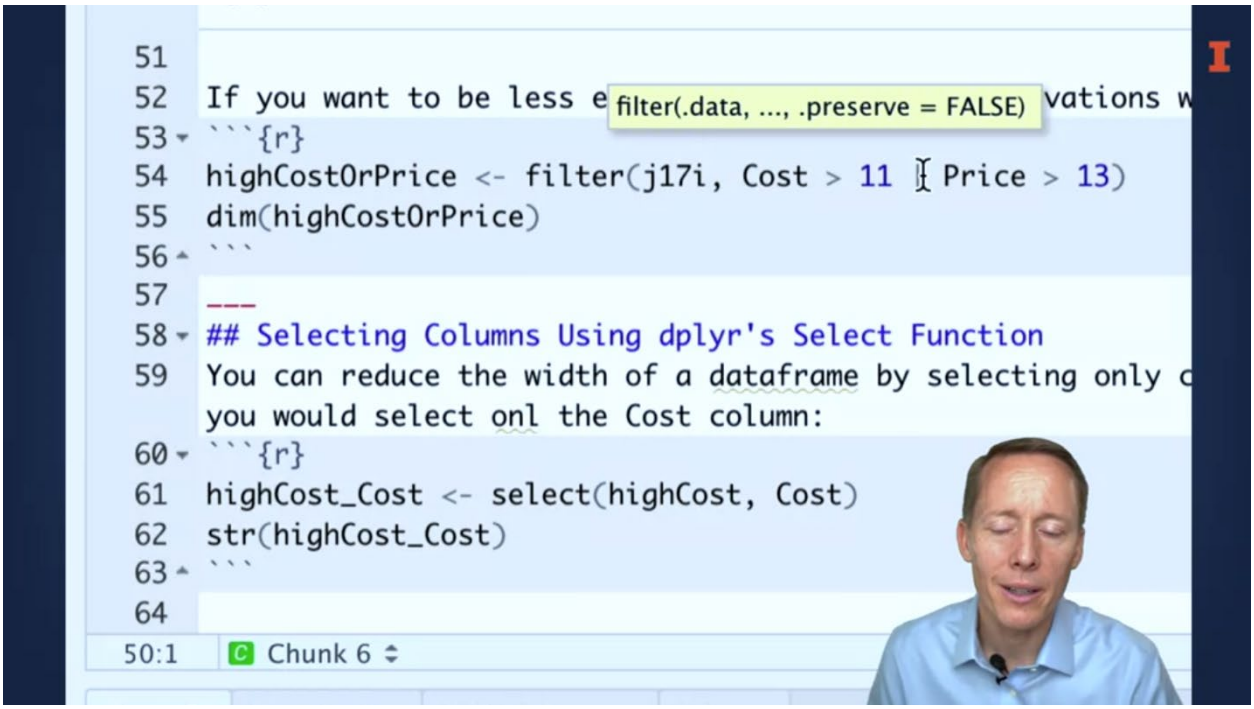


And we can also use the dim function to tell us that we've got 14 rows and 21 observations. Now I don't have to use the filter function from the dplyr package. I could do this using base R in the following way. I'll indicate the name of the data frame, and then I'll use square brackets and indicate the column of that data frame, and say all values that are greater than 11, and then a comma with nothing else other than the closing square bracket to indicate I want all the columns. So if I do this and save it to no

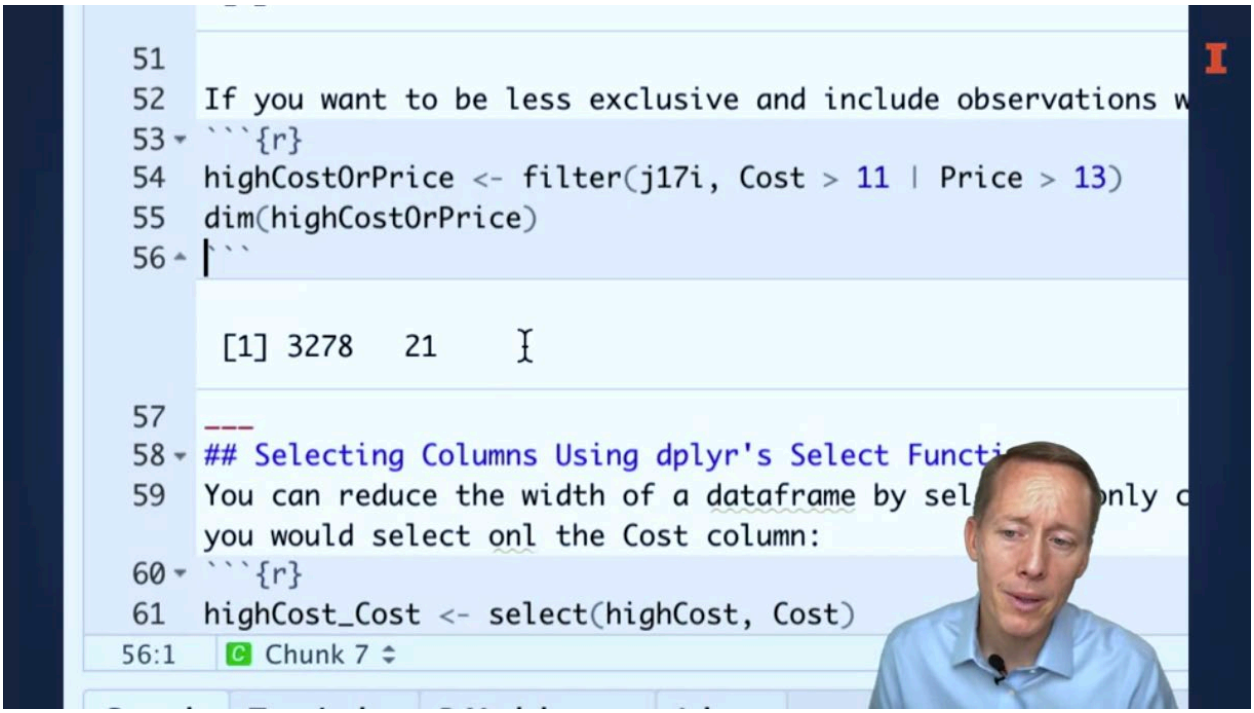
object high cost base, you can see that and the global environment it's the same number of rows and columns as also indicated using the dim function.



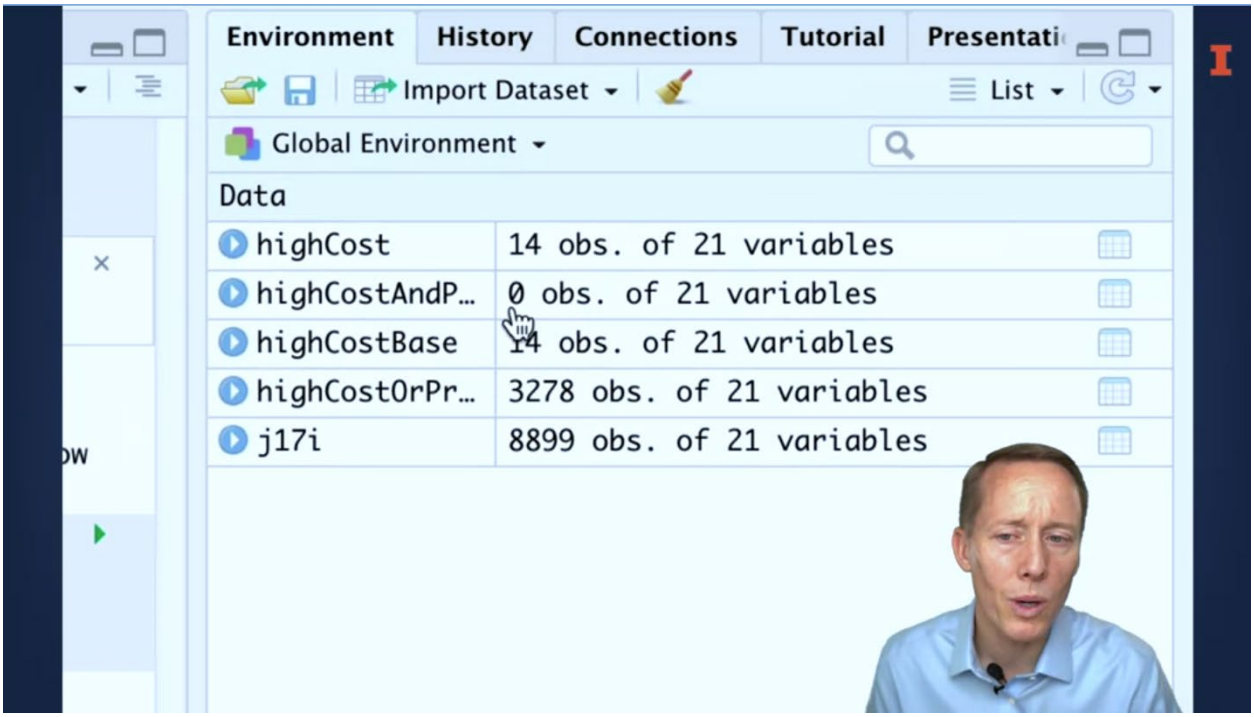
Notice that the base R function requires you to type out the name of the data frame twice. Now, in this example, it's not that big of a deal. But when you start performing many preprocessing tasks, they add up and it gets very cumbersome, all right.

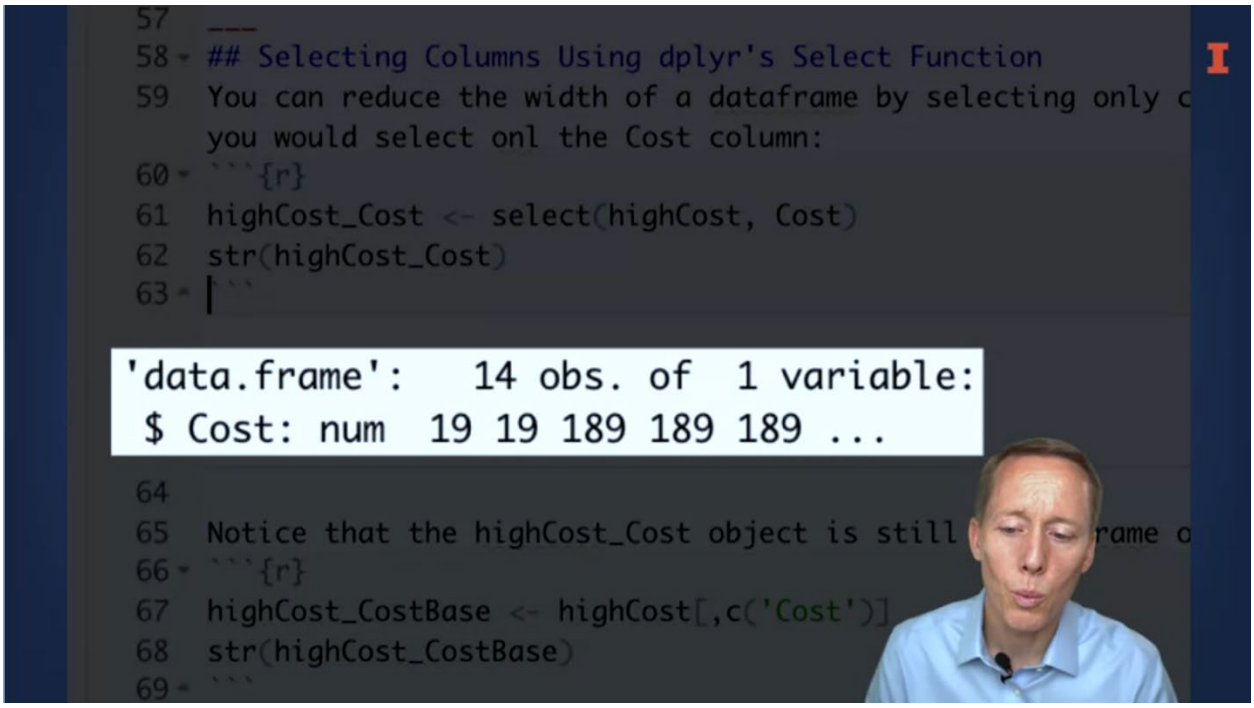


Now the filter function allows you to filter the number of rows based on more than one criteria. For instance, if I want to narrow down the number of rows only to those observations where the cost is greater than 11 and the price is greater than 13, I can very easily do that by modifying that initial function, by just including the ampersand sign along with the column name, price and the criteria, it needs to be greater than 13. If I run, look at the dimensions, there's no observation that meets those criteria. Now, I can also be less exclusive and filter down the number of rows to those observations where either the cost is greater than 11 or the price is greater than 13. So that's an or, rather than an and argument there. The way we do that in R is using the pipe symbol. So I'll include that in this filter command rather than the ampersand sign and run that, and look at the dimensions.

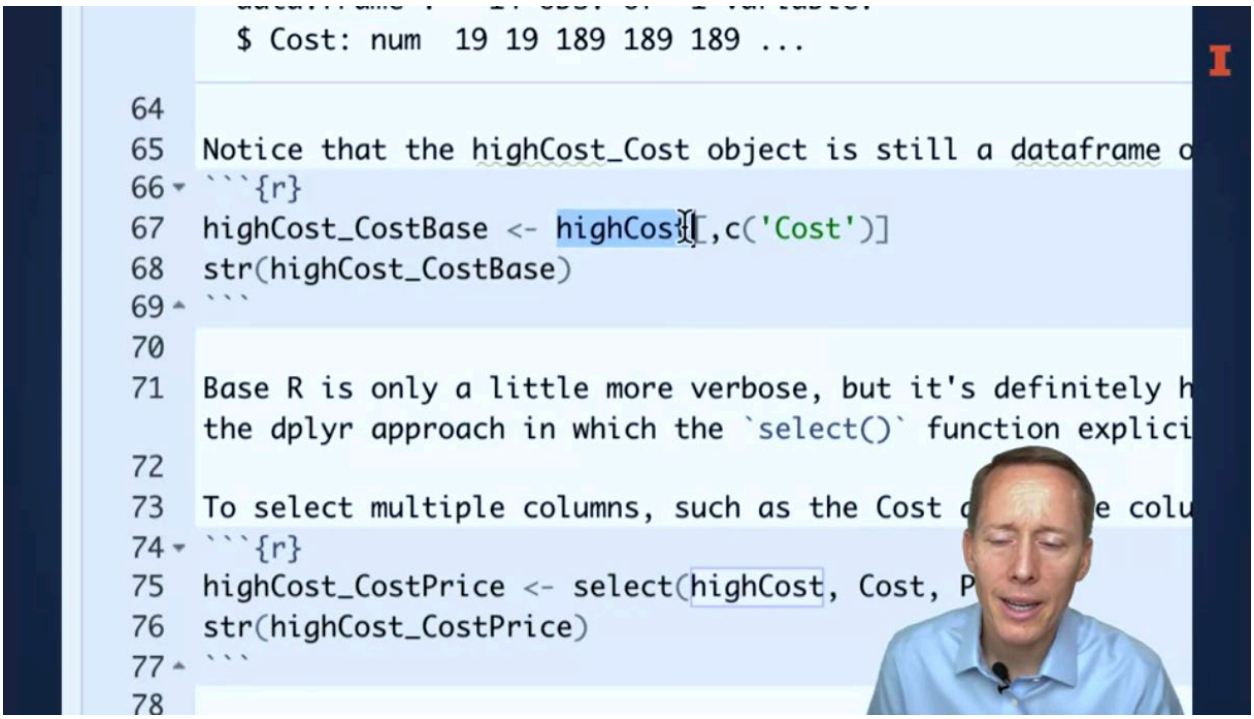


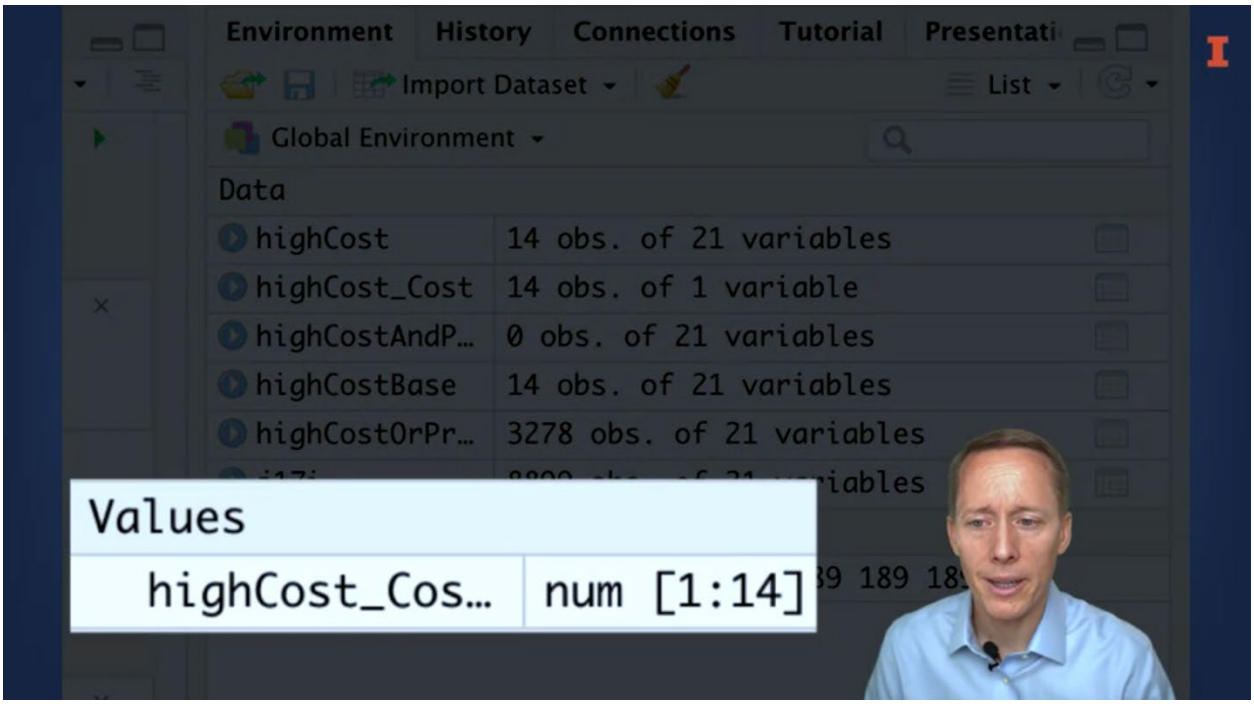
And this time it is less exclusive, and so it leaves us with 3278 observations. All right, so there's an example of how you can use the dplyr filter function for reducing the number of rows in a data frame. Let's talk about now how you can reduce the number of columns of a data frame. So let's say, for instance, we want to select only the cost column from the high cost data frame that we created above.



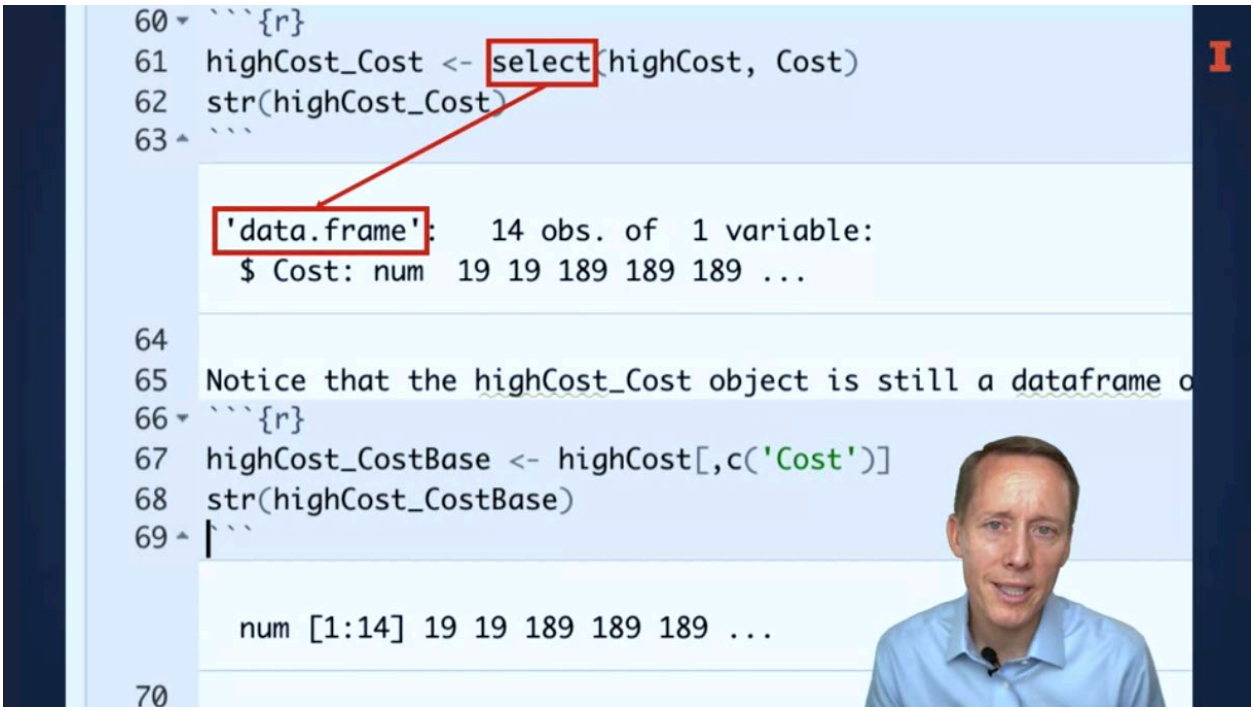


You can see that this data frame currently has 21 variables. So I will use the select function and then in parentheses. I'll first indicate that I'm going to use this function on the high cost data frame, and then the column of that data frame is cost. All right. And now if I evaluate the new data frame, highCost_Cost, you can see that it's 14 observations with one variable, and that variable is cost.

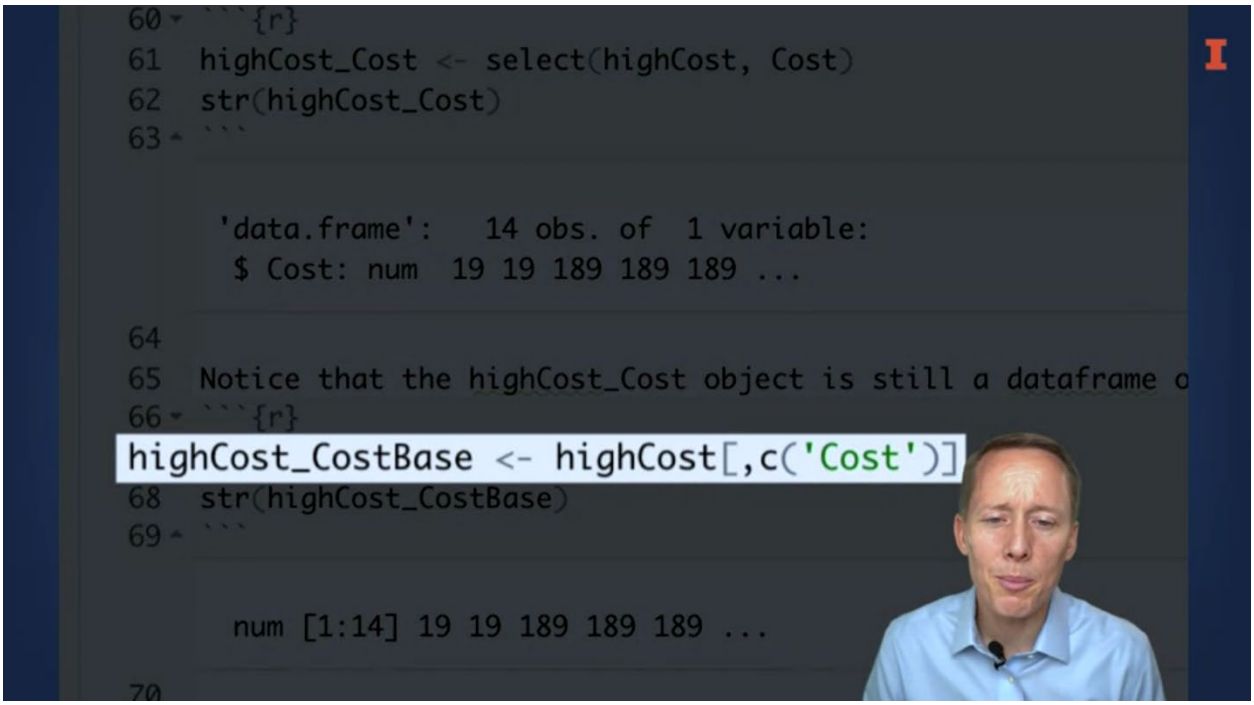




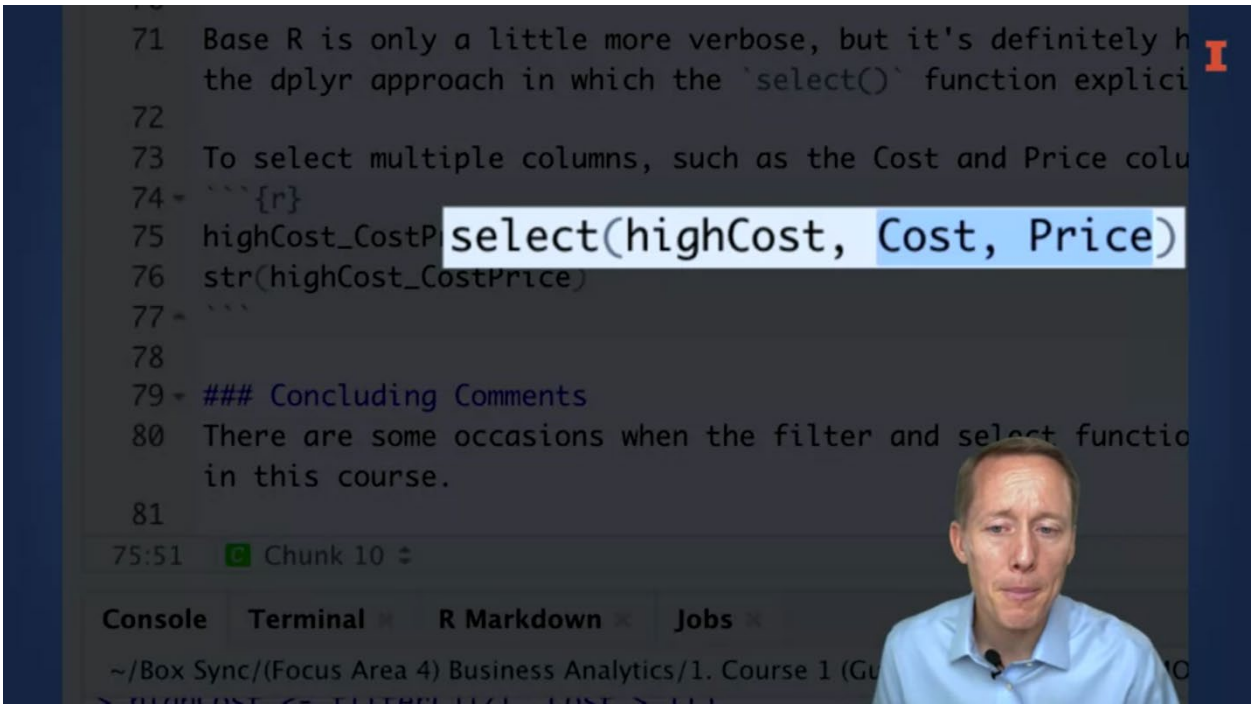
Let's compare that to the base our approach, where I can take that high cost data frame and reduce the number of columns by using the square brackets and then a comma to indicate I want all the rows. And then I need to combine the names of all the columns, and I need to put the column names in quotation marks. So if I do that here, I get this new highCost_CostBase object, and notice that it is not a data frame, it is a vector. So that's one difference of using base R to filter down columns of a data frame versus select from the dplyr package.



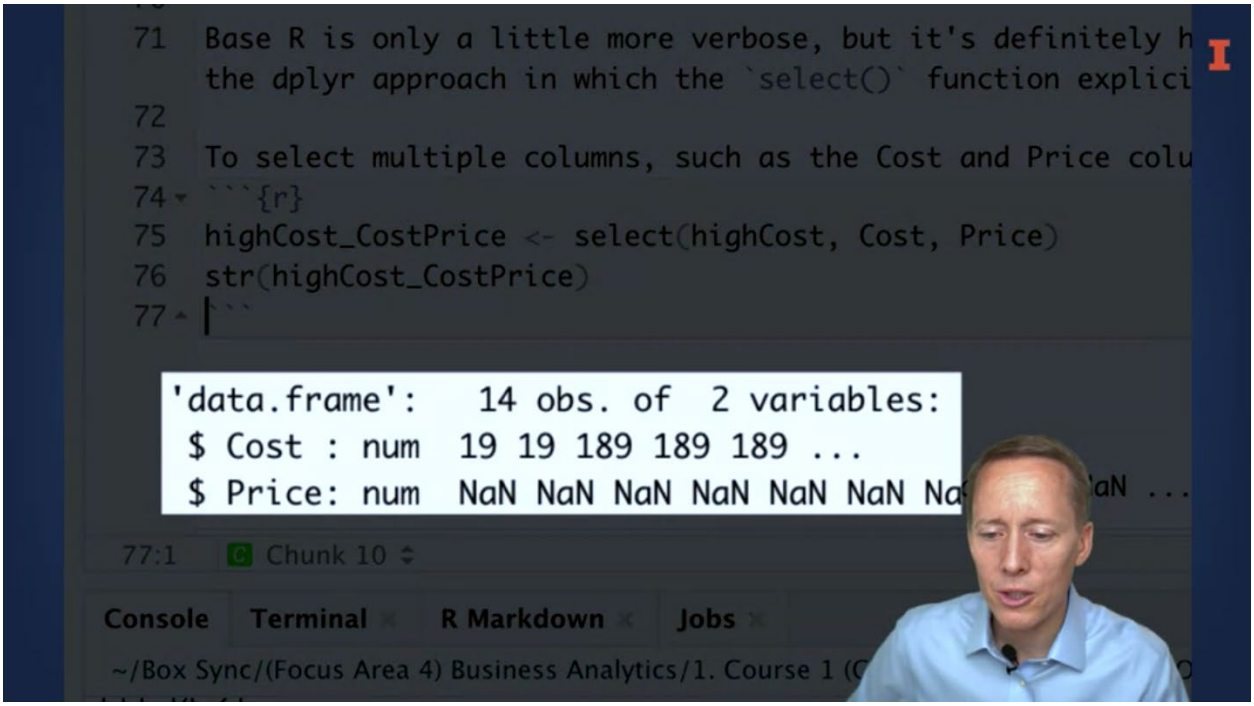
Select will retain the data frame object structure.



So base R is only a little bit more verbose, but it's definitely harder to read. It takes more mental power, I think, to identify what's going on compared to the dplyr approach, in which the select function explicitly indicates that you're selecting certain columns.



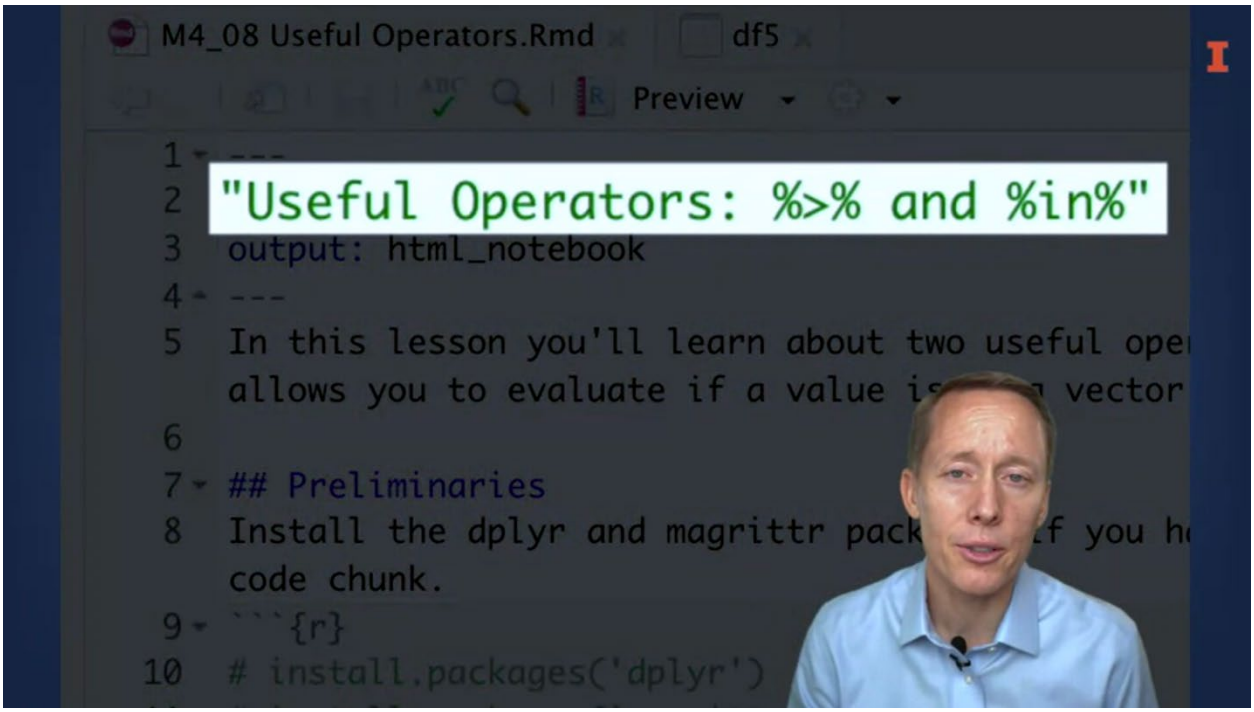
Now what if you want to select more than one column? You can easily do that using the select function, and all you do is you separate those column names by a comma. And again notice that you don't have to include quotation marks around the column name. So let's go ahead and create this new data frame that only has cost and price in it. And I'll look at the structure of that data frame.



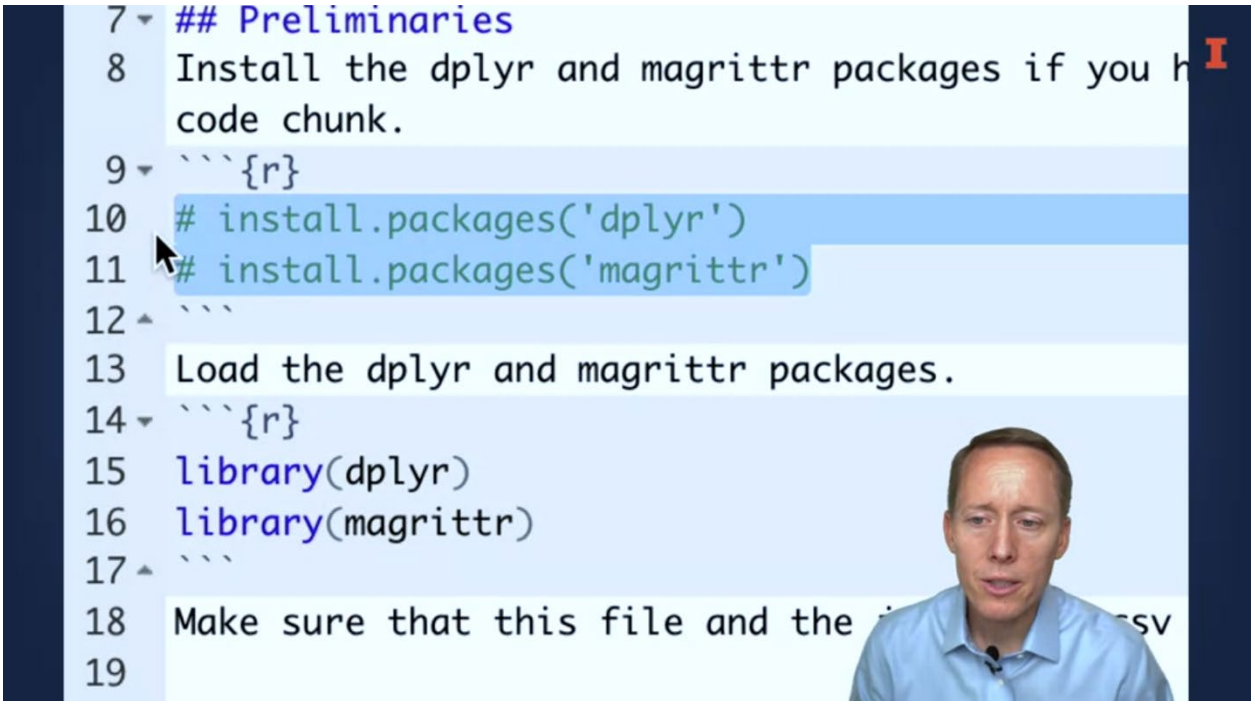
And sure enough, we've got a data frame with 14 observations and only 2 variables,

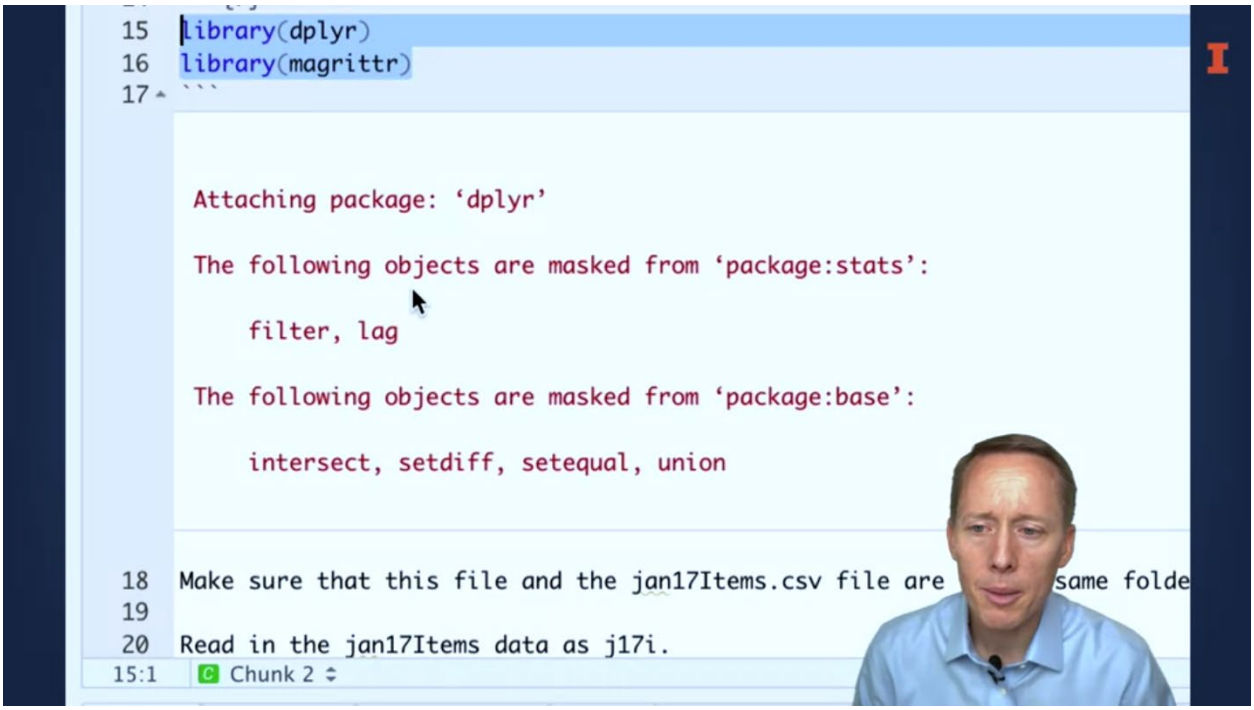
cost and price. So there are some occasions when the select and filter functions do not allow you to do what you want to do, and so you have to use the base R functions. But at this point, you don't need to worry about those.

Lesson 4-1.8 Useful Operators: %.% and %in%



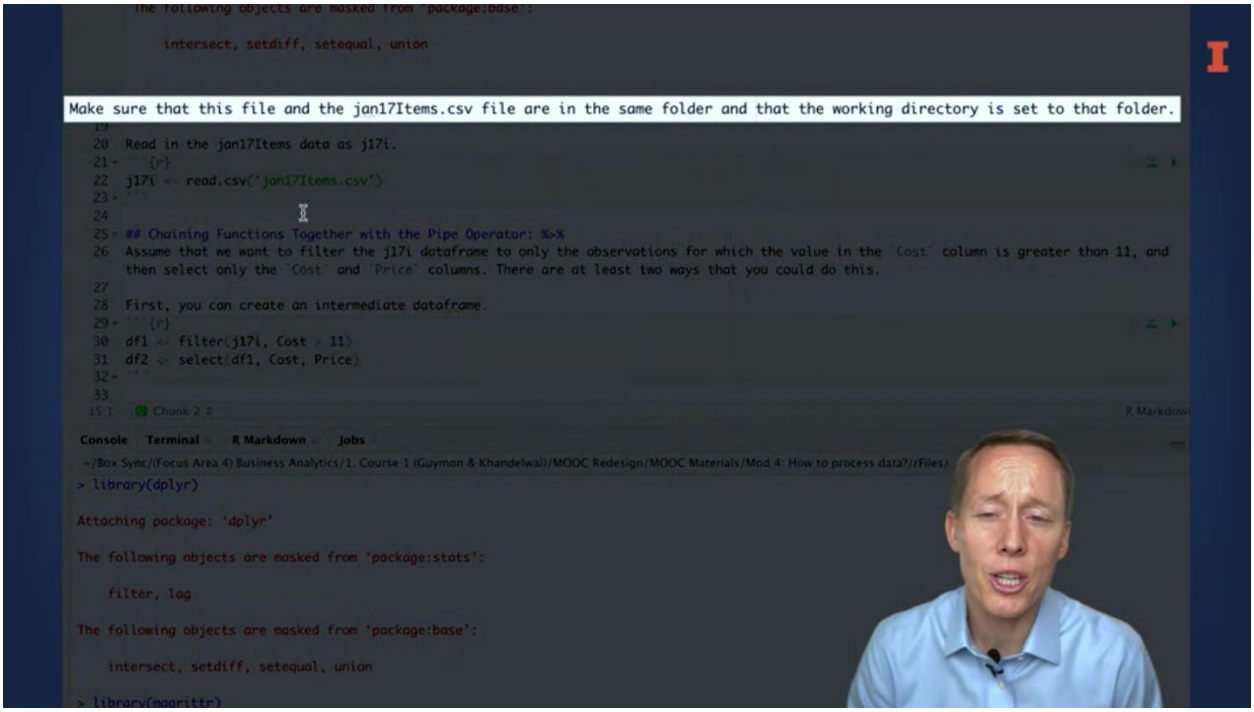
In this lesson, I want to introduce you to two very useful operators for making your code more efficient and easier to read and write.



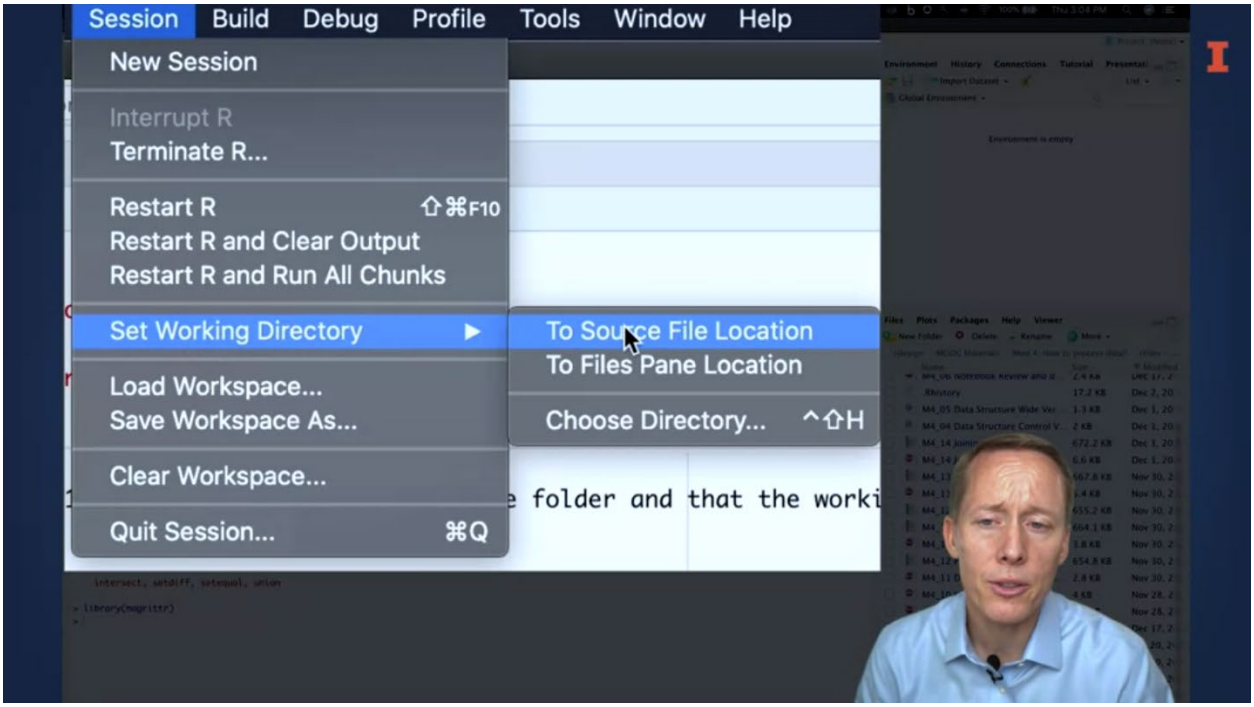


We are going to be using the dplyr and magrittr packages, so if you have not installed those packages, please make sure and do that. Make sure and run the library function on both of those packages so that you can access the functions in those packages. Notice the warning here. It says, after attaching the dplyr package there are some objects masked from the stats and base package, specifically the filter, lag, intersect, setdiff, setequal, and union. What that means is that, both the dplyr and the stats package contain a filter and lag function. If you were to run the filter function without

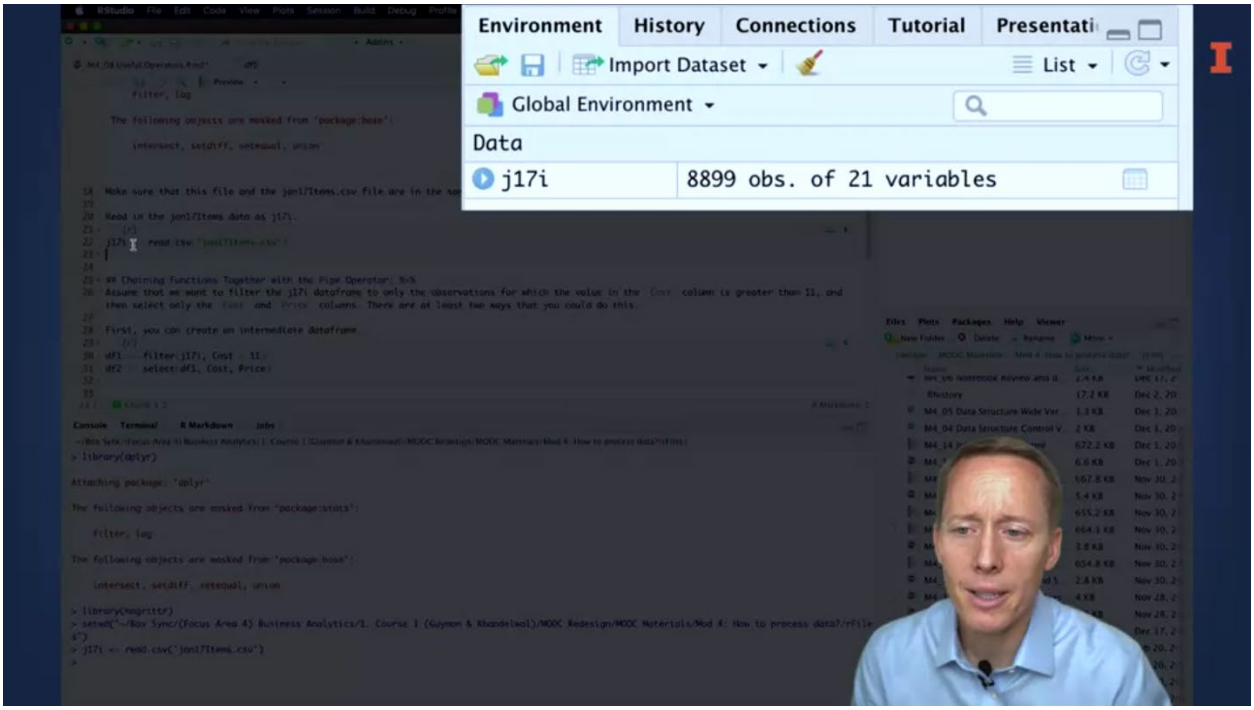
referring to either package, it will assume you're referring to the dplyr package, it's masking the stats package.



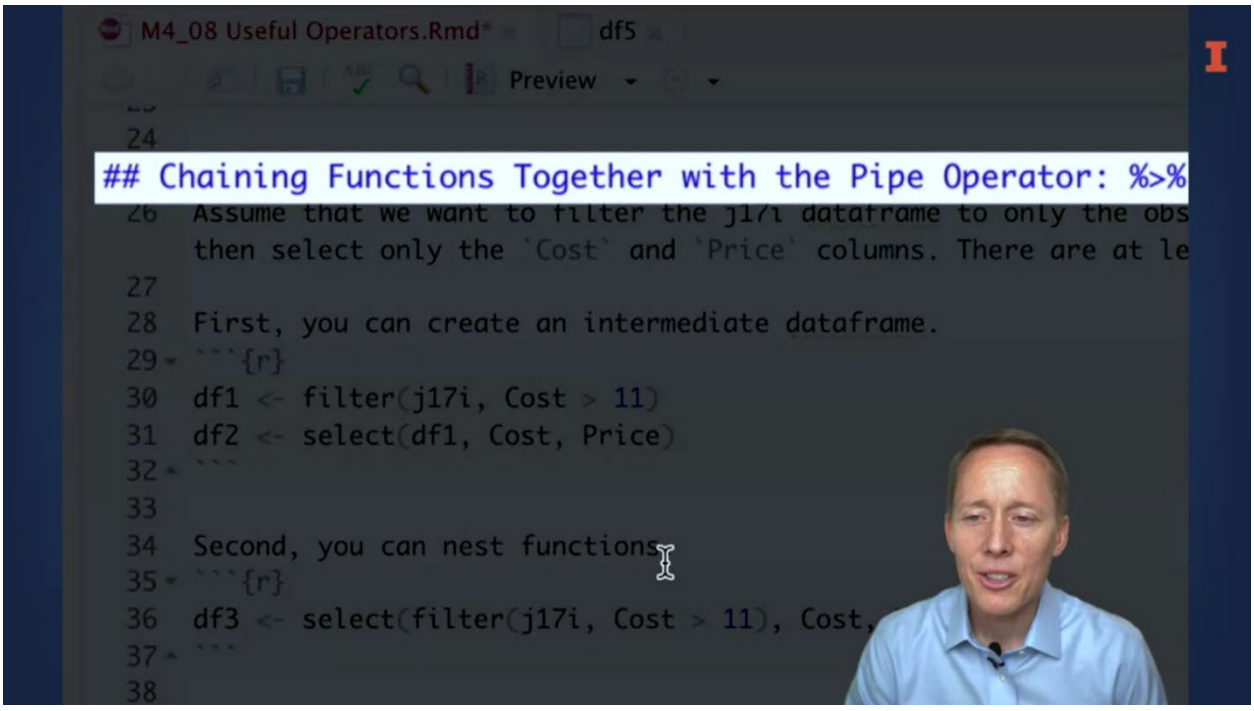
Make sure that this file, our notebook file and the Jan17items.csv file are in the same folder, and that the working directory is set to that folder.



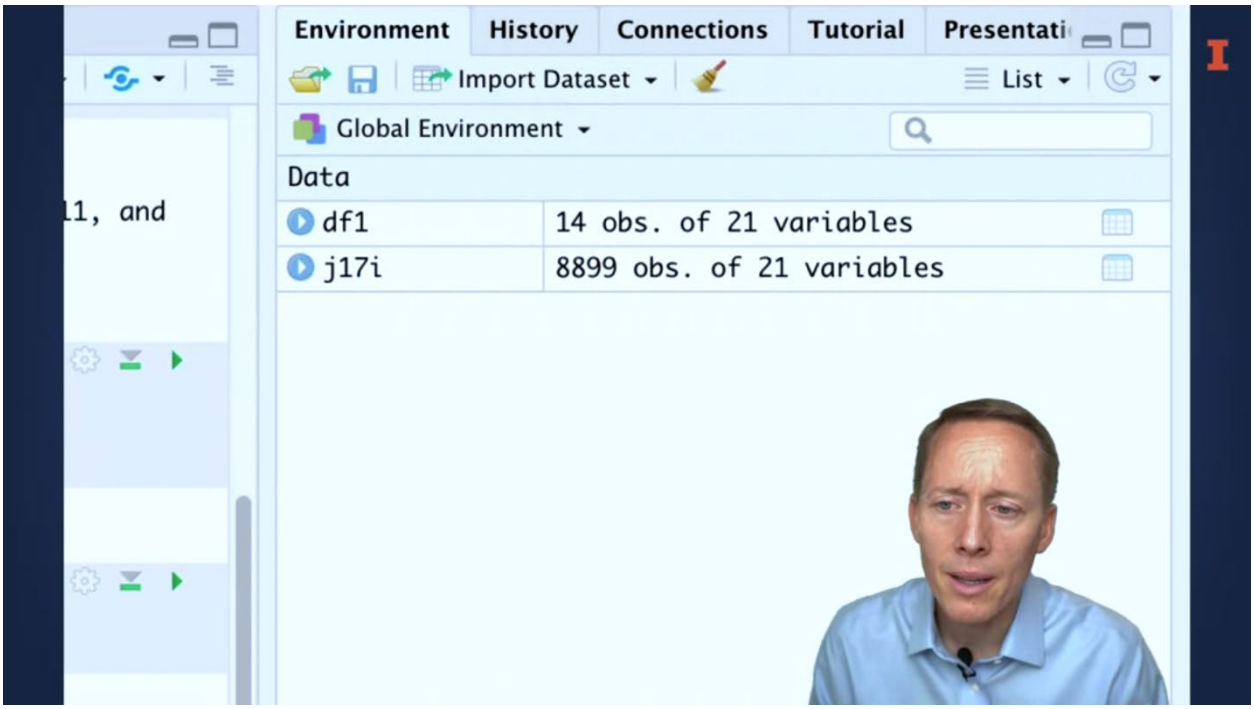
As a reminder, you can set the working directory by going to the Session menu item, go down the Set working Directory and then to Source File Location.

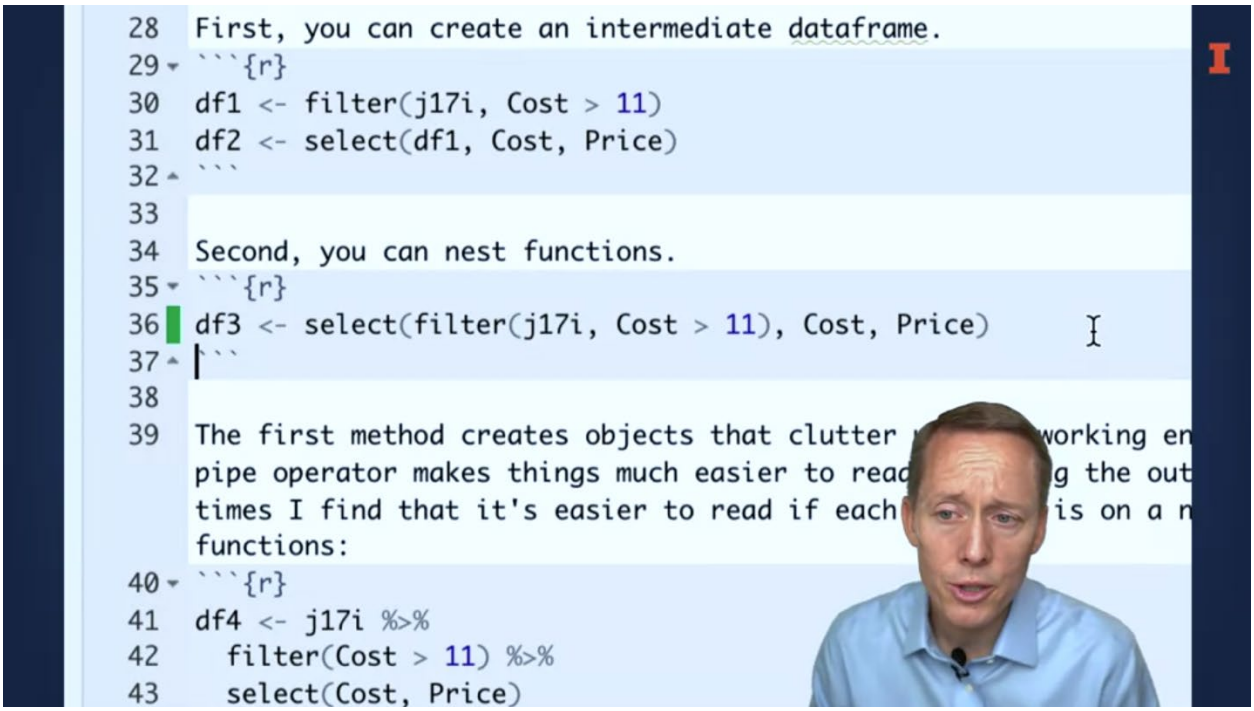


All right. Go ahead and Read in the Jan17Items.csv File and create a new Data Frame object called j17i.



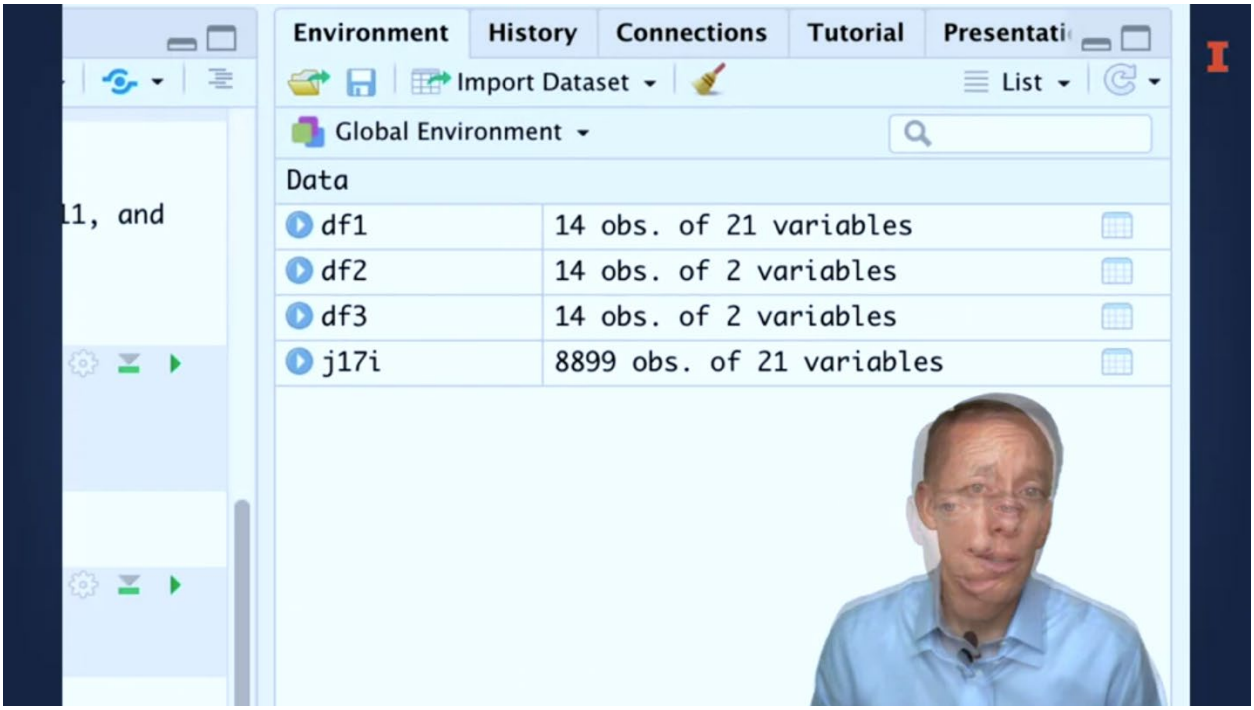
Let's talk about how we can use the Pipe Operator to chain functions together. Assume that we want to filter the j17i data frame object to only the observations for which the value and the cost column is greater than 11, and only to the cost and price columns. I want to make it shorter and skinnier. There are at least two ways that you could do this without using the Pipe Operator.



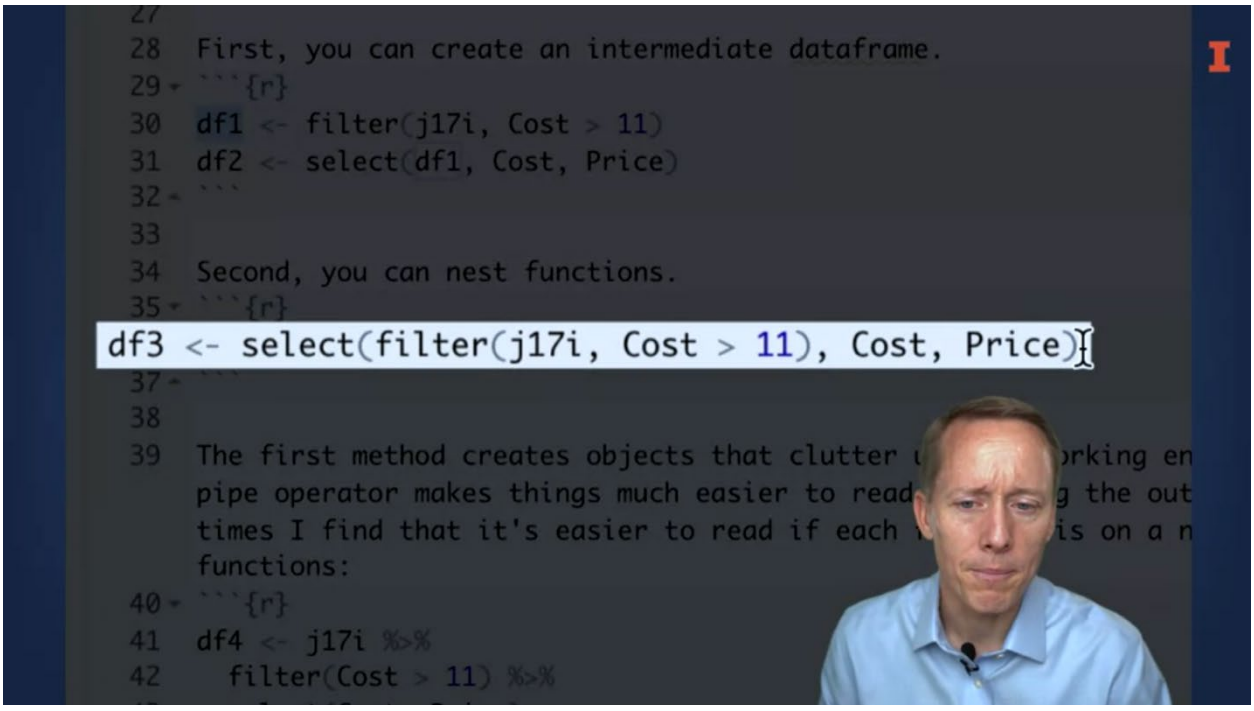


First, you can create an intermediate data frame. What I mean by that is that you would use the filter function on the j17i data frame object, and filter the observations down to only those for which cost is greater than 11, and store that in a new data frame object df1. We can look and see that there's only 14 instead of 8,899 observations, so it is shorter. Then we'll take that df1 data frame object and put it into the Select Function. We'll indicate that we only want to keep the cost and price column. If we run that, we've got df2, which only has two variables instead of 21. There's one approach. Another

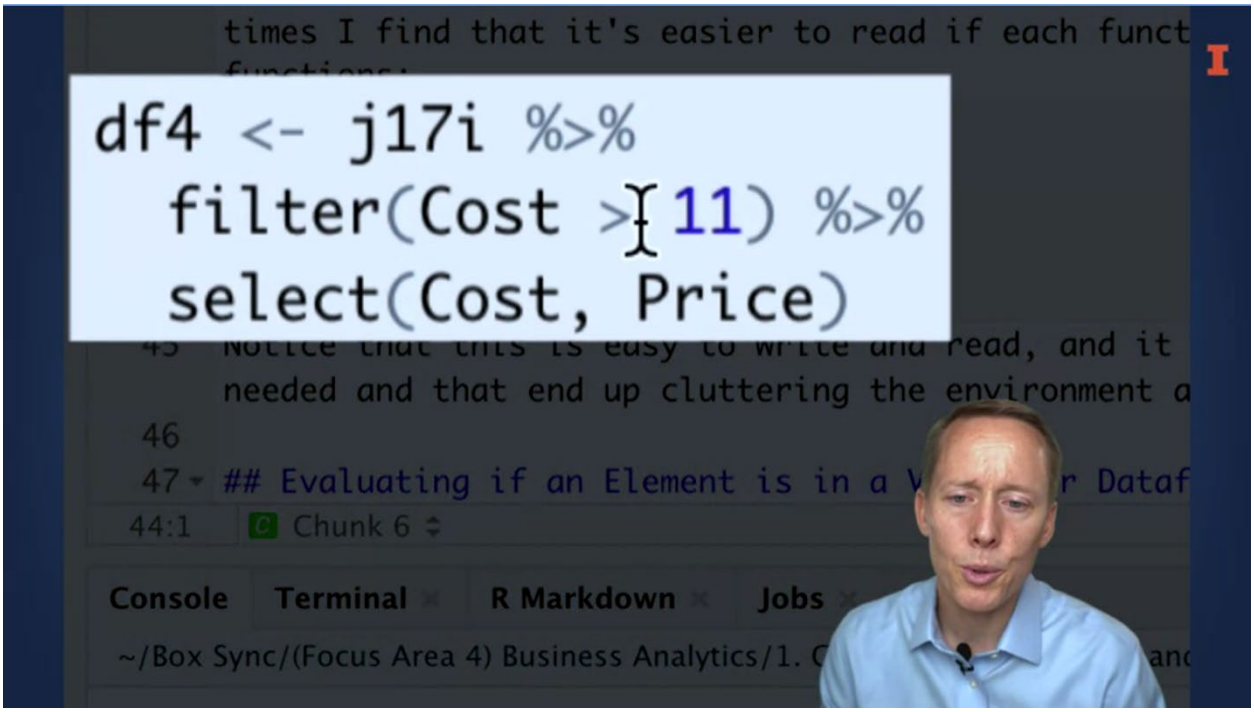
approach is that you can nest functions together. We can take this Filter Function right here and wrap it in the Select Function. That should result in the same thing.



Sure enough, we get 14 observations and two variables. Both of those approaches work, but the first approach creates an intermediate data frame object which is inefficient. You'd have to delete that at some point, especially if you're dealing with large data frames.

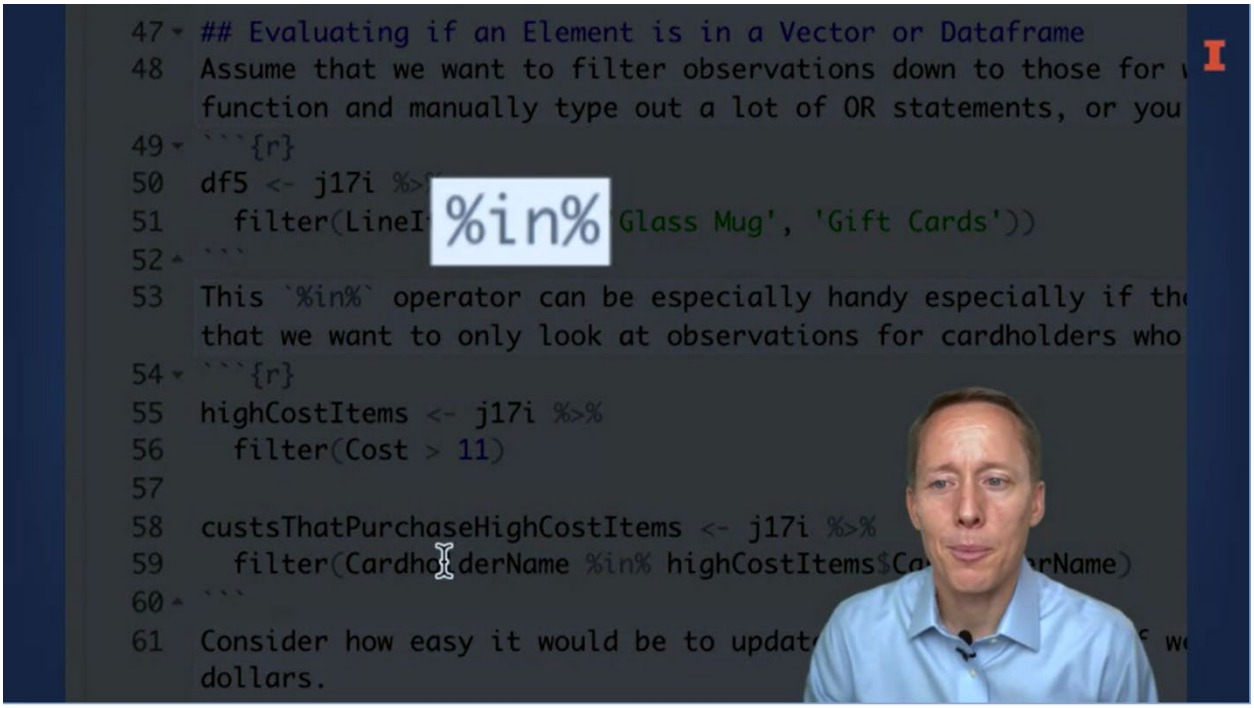


The second approach is just hard to read. This is where the Pipe Operator comes into play.

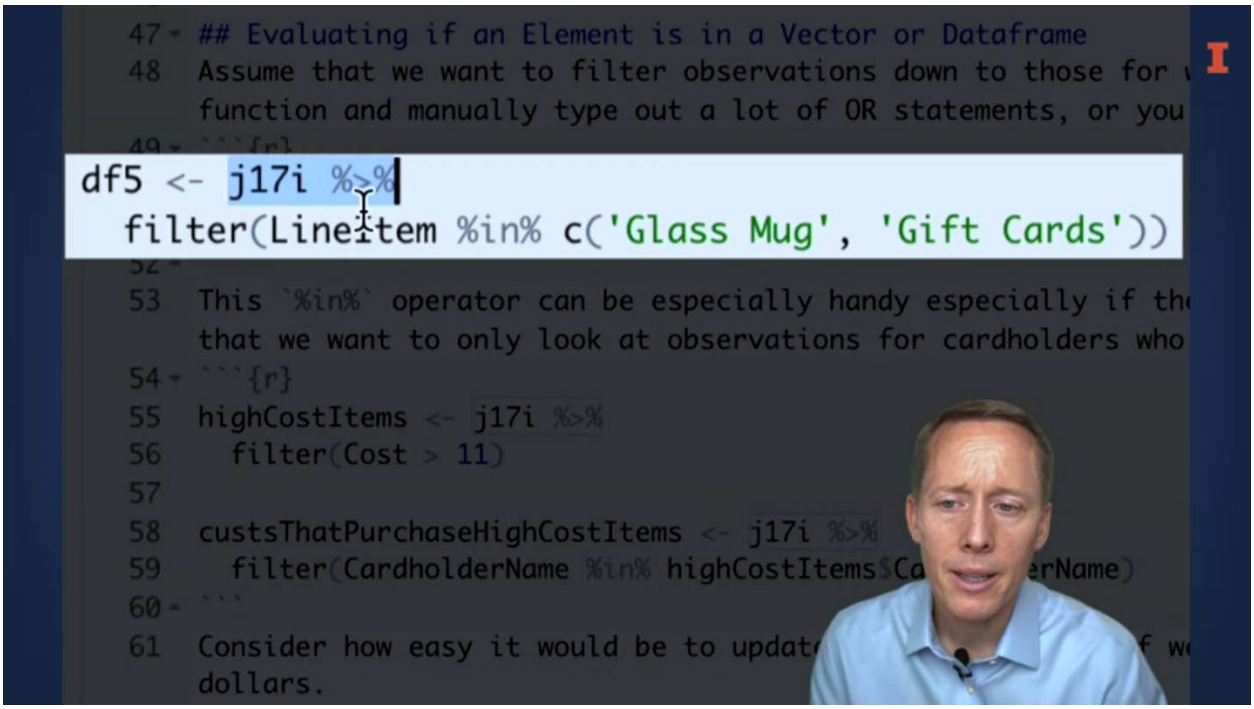


Let's perform the same task using the Pipe Operator. The pipe operator is this percent greater than percent symbol all next to each other. You can think of it as taking what's on the left side and piping it into or sending it into the next function. Oftentimes, to make it easier to read, I'll put each function on its own row. Let me explain what's going on

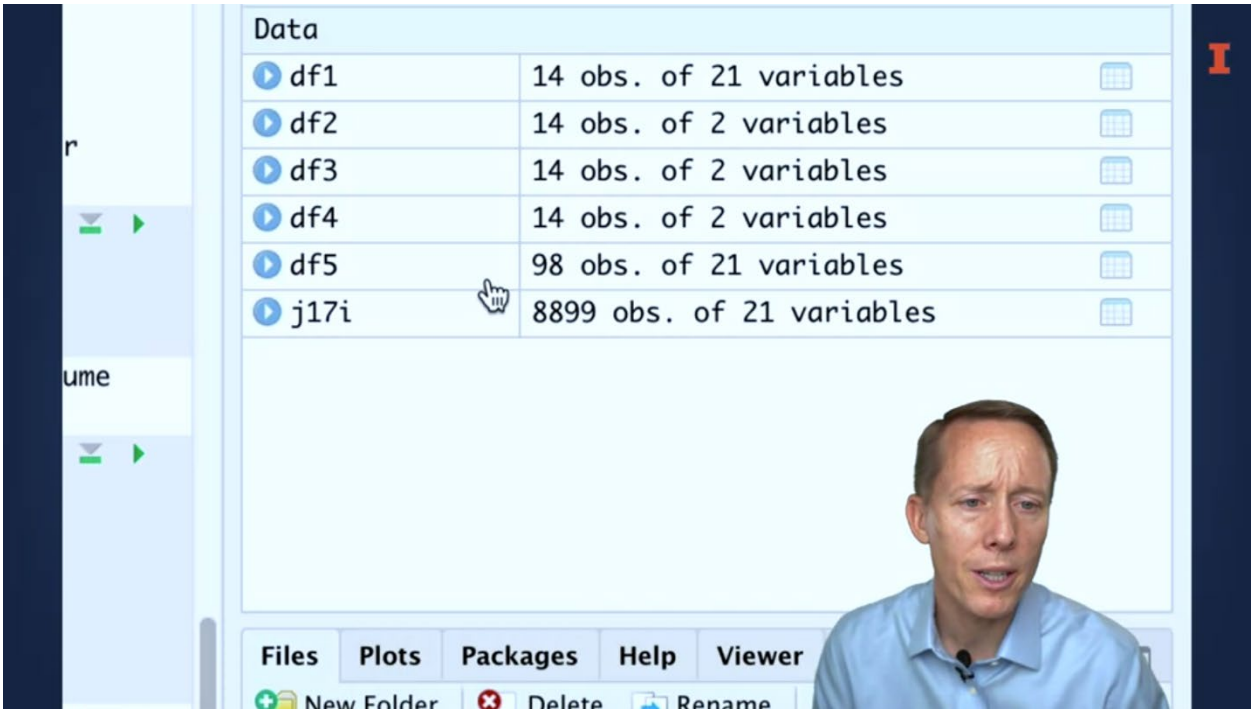
right here. We're taking the j17i data frame object, sending that into the filter function. Notice that we don't have to retype out j17i. It knows that it's already being inserted in there. Then we're filtering it to observations which cost is greater than 11, and then we're going to take that intermediate data frame object, send it directly into the select function where we identified that we only want to keep the cost and price columns. If I run that, we can look and see that we've got the df4 data frame object, and it's the same as df2 and df3 there. It is the same dimensions. If you were to compare it, it would be exactly the same. Notice how much easier this is to write and read. It takes fewer characters, oftentimes as well. It also prevents the creation of additional intermediate data frame objects that end up cluttering the environment and may take up additional memory as well.



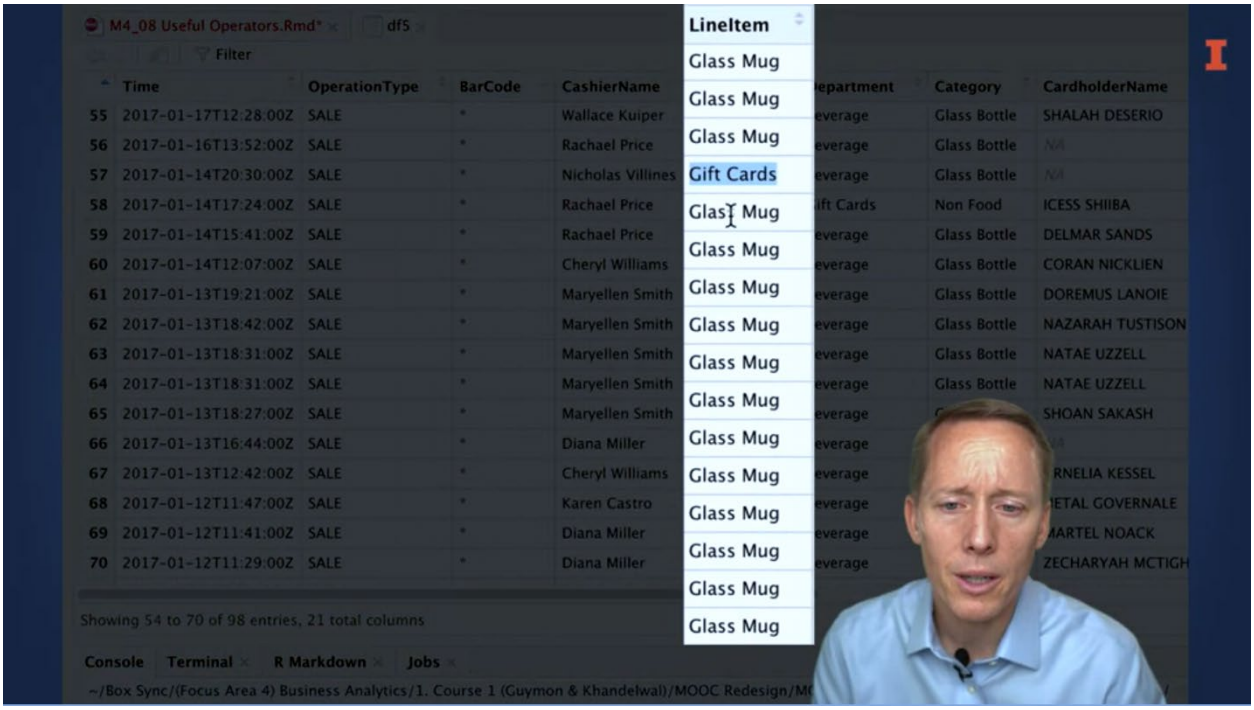
Let's talk about using the %in% operator. The %in% operator allows you to test if a value is in a vector of other values. It makes it so that you don't have to type out a filter statement that includes lots of [inaudible] in it. Here's how we could use this in a business analytics setting.



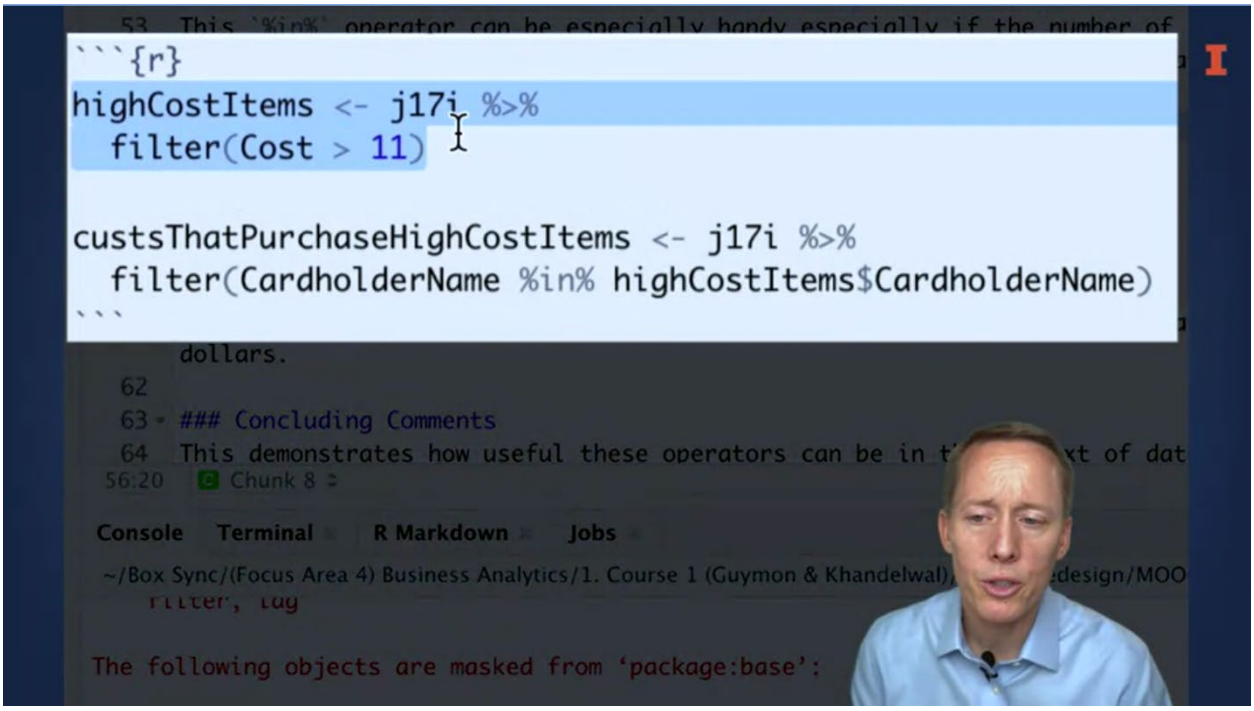
Let's assume that we want to filter the j17i DataFrame object to only those rows for which the [inaudible] is equal to, either a glass, mug, or gift cards. I'll also use a Pipe Operator here, we'll take that j17i, put it into the filter function. We're indicating that, "Hey, we're going to look and keep only those observations for which LineItem is in," so this is the %in% operator, this vector, which is made up of glass, mug, and gift cards.



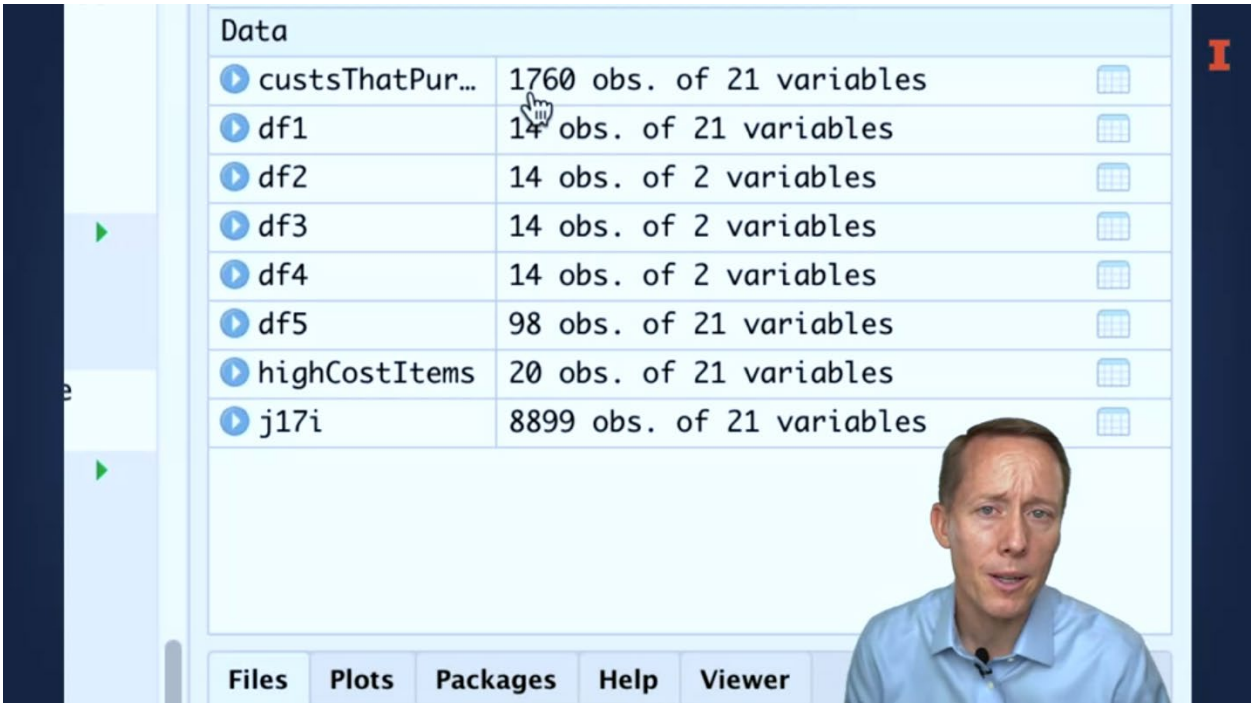
When I do that, if I run that, I've got this new DataFrame object that has 98 observations and 21 variables.



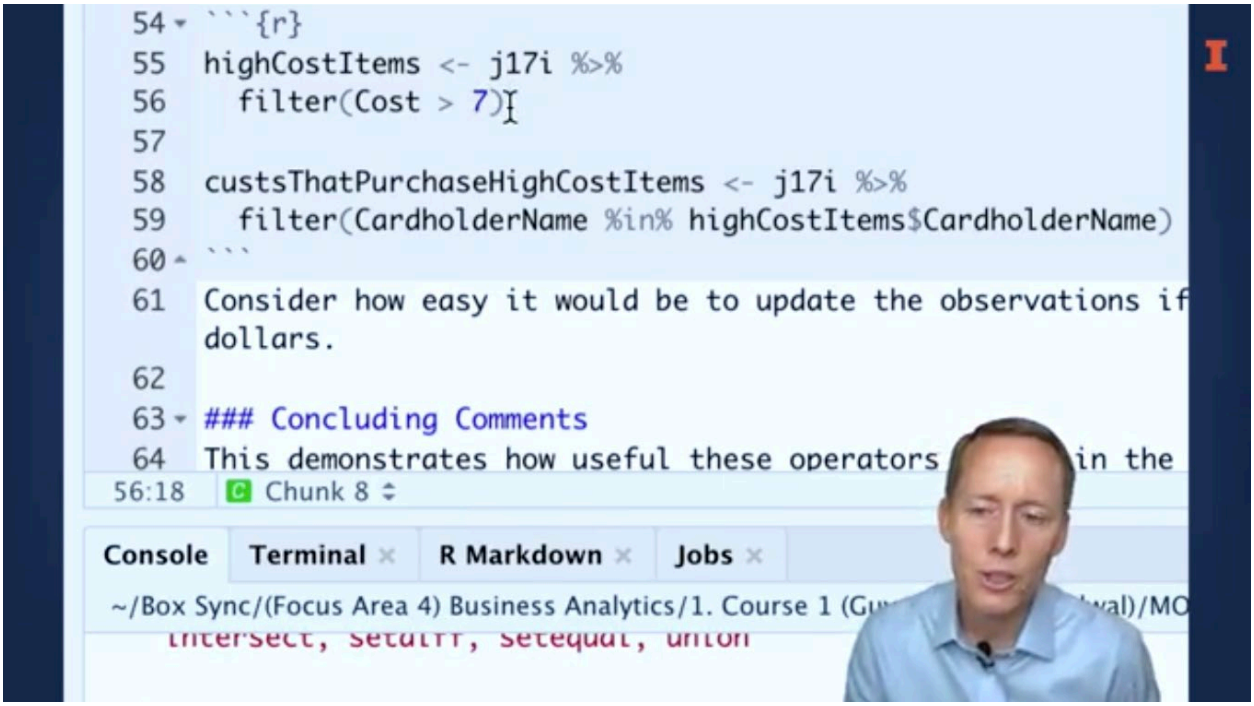
If I look at the LineItem, I can quickly evaluate. Just visually inspecting, it looks like, sure enough, most of it is glass, mugs, there are at least one gift card in there as well. There's another.



That %in% operator is especially handy if the number of items in a list is long or dynamic. If it's changing. Let's say for instance, that we want to look only at observations for cardholders who purchased at least one high-cost item. Here's how we could do that. We'll create a DataFrame for which the cost is greater than 11, and we'll call that, High Cost Items. We'll create another DataFrame, ThePurchased HighCostItems, set that equal to j17i, and then filter it such that the CardHolderNames have to be in this high cost items DataFrame and in the cardholder name column.



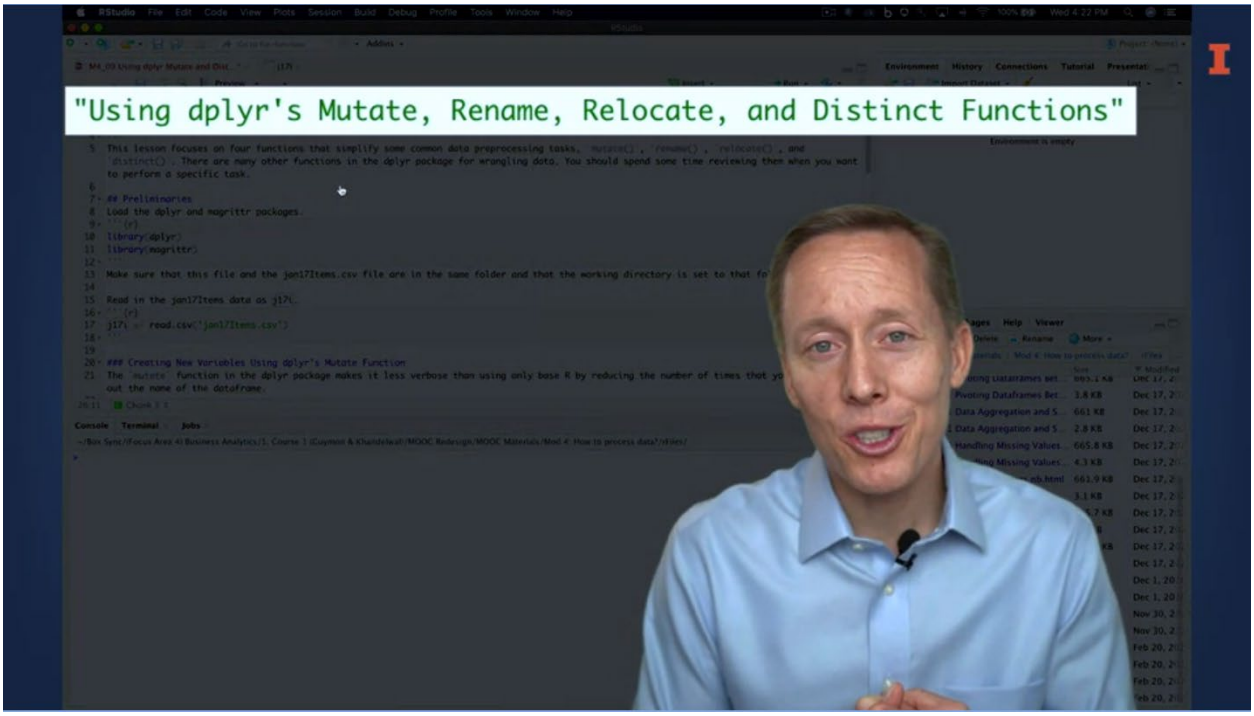
When I do that, I get this new DataFrame object that has 1,740 observations. If you were to look at the names and the cardholder names column, that new DataFrame, they'd be the same ones as in the high cost items DataFrame, but you would have all of their transactions.



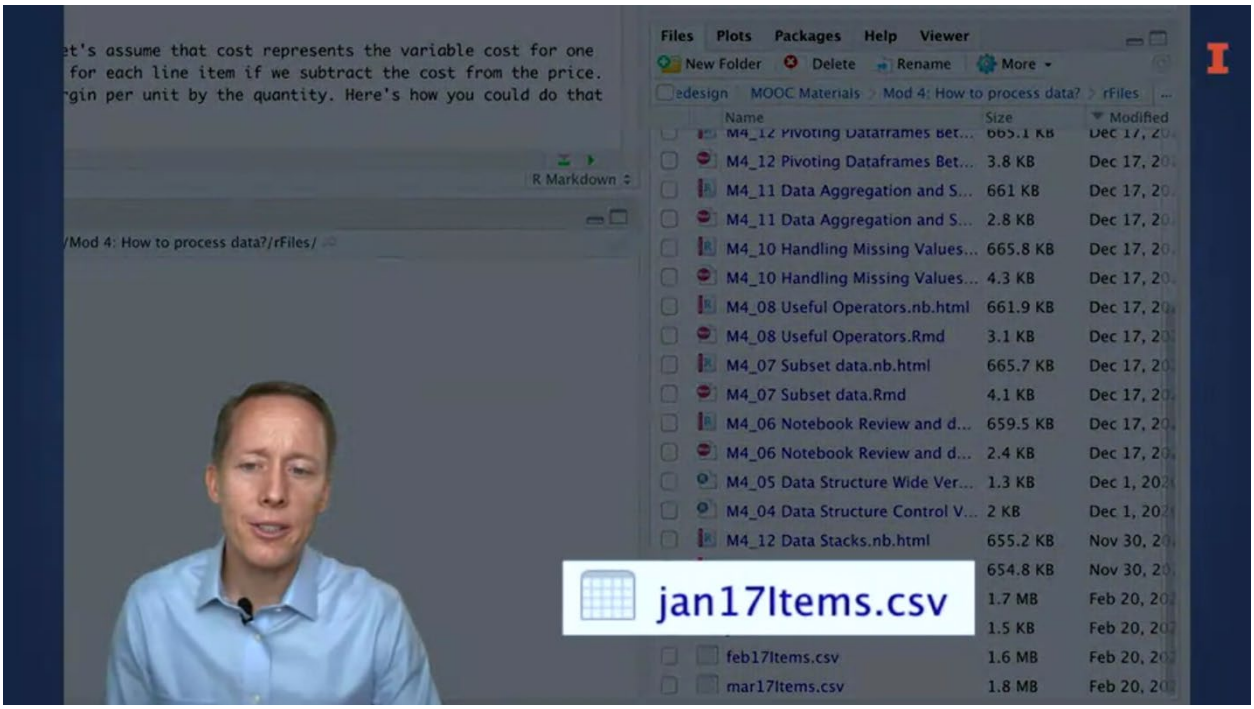
Consider how useful this is because we don't have to type out individually each of those line items in there. We can also update it very quickly. Lets say, for instance, we think

high cost items are those for which the cost is greater than seven, we just changed that 11 to a seven, then run both of those and now we've got 1,760 observations. The Pipe Operator and the %in% operator are very useful. We'll be using the pipe operator in almost every other lesson from here on out. I hope this gives you an idea of why they're useful and how these operators make your code easier to read, write and more efficient.

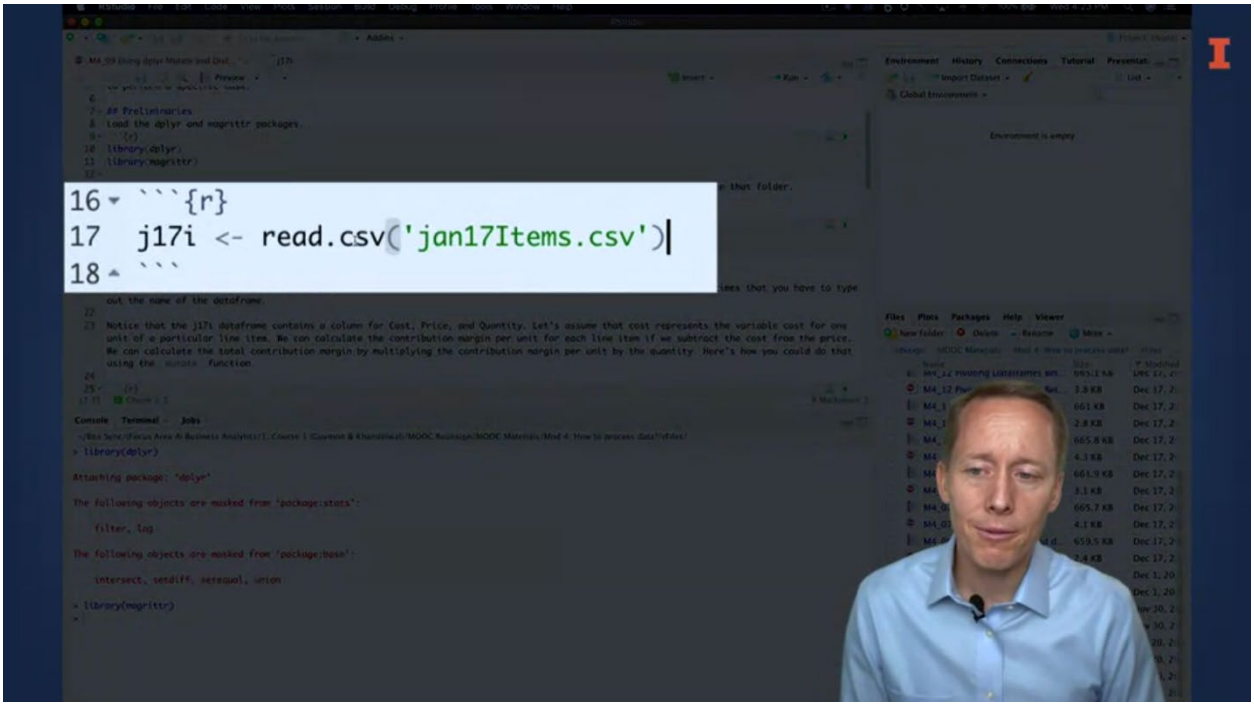
Lesson 4-1.9 Using dplyr's Mutate, Rename, Relocate, and Distinct Functions

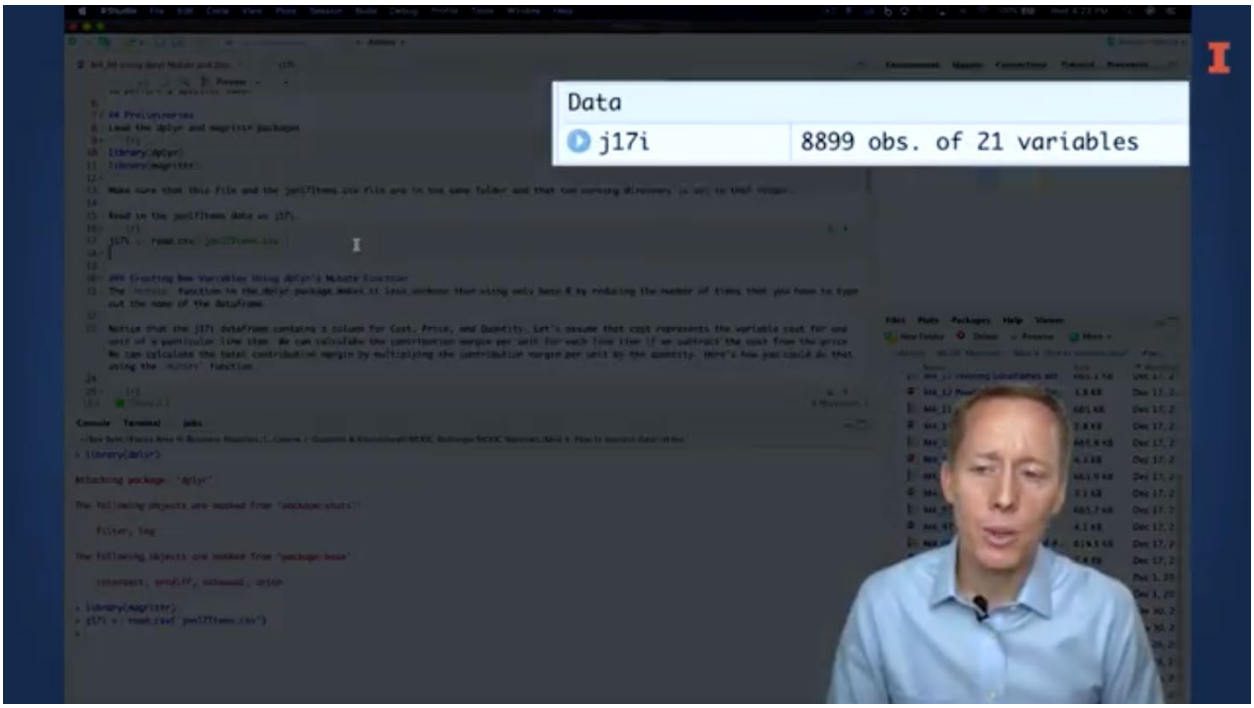


In this lesson, I want to introduce you to four very useful functions from the dplyr package for pre-processing data, the Mutate function for creating new columns, the rename function for renaming columns, the relocate function for rearranging the order of columns, and the distinct function for calculating the unique or distinct values of a column of data.

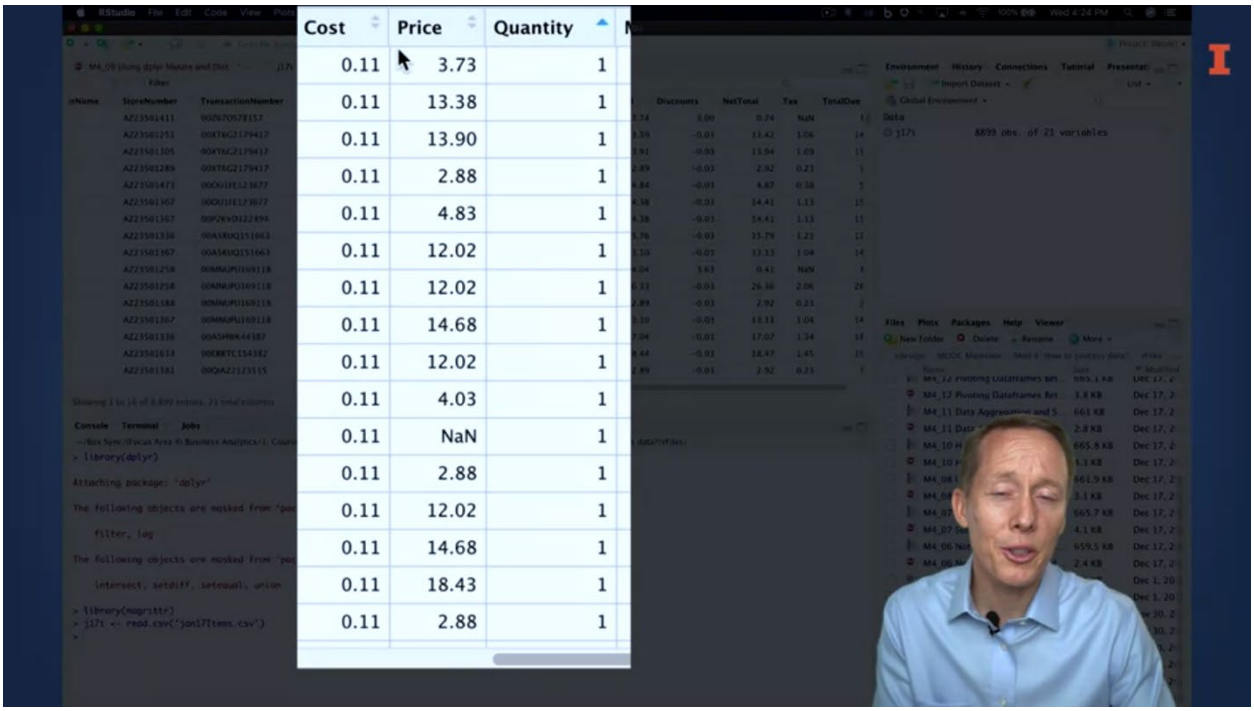


So first, make sure that you have installed and loaded the dplyr and magrittr packages. Also, if you're following along, make sure that this, our notebook file is in the same folder as the jan17Items.csv file, and that the working directory is set to that folder.

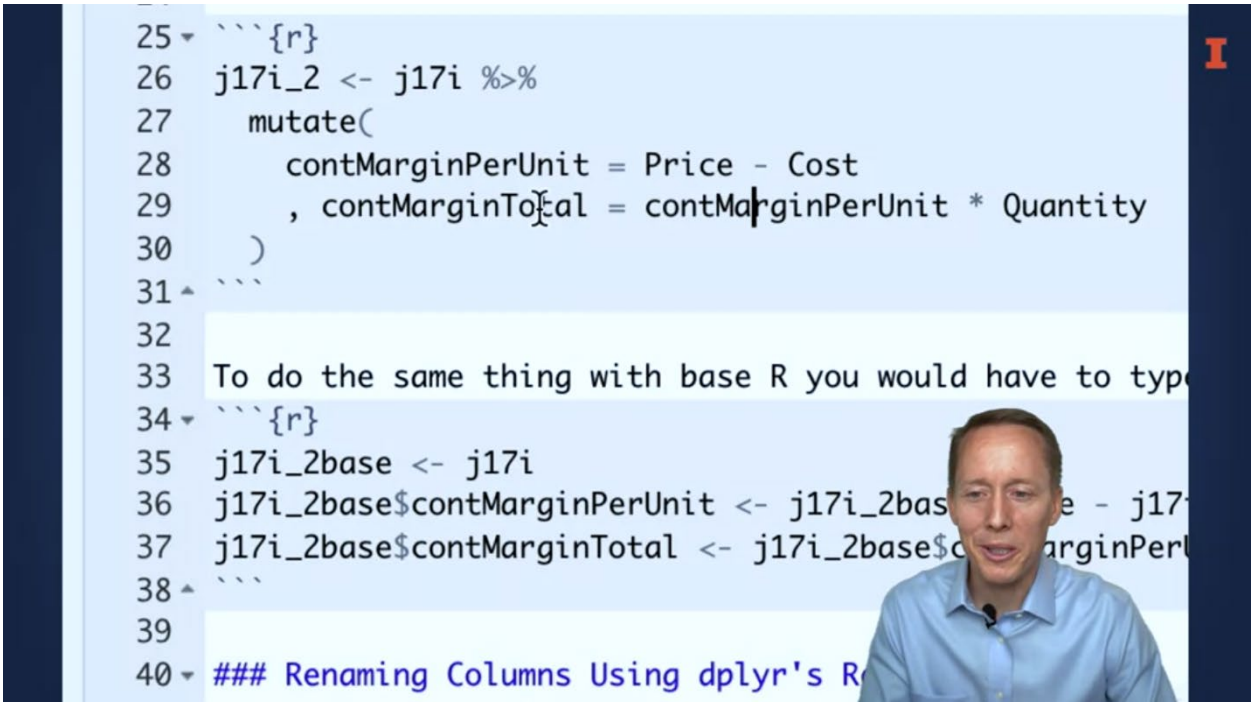




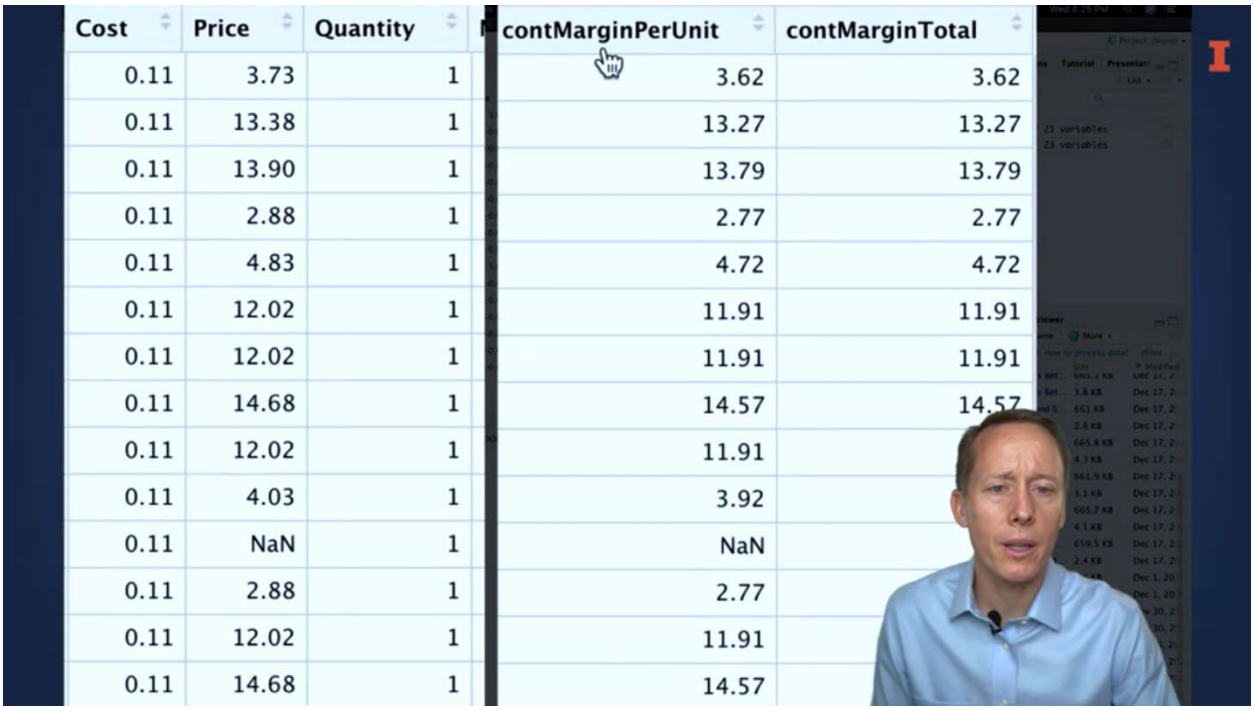
Once you've done that, you can read in the jan17Items.csv file and create a data frame object, which I have named, j17i. Now let's talk about the mutate function. So oftentimes a pre-processing task is that you want to create calculated fields.



For instance, if we look at the j17i data frame object, we can see that there is a cost column, a price column, and a quantity column. Let's say we want to calculate the contribution margin in total for the line item, for every line item here. So we could do that in two steps, and the way we would do that is maybe calculate the contribution margin per unit, calculate that as a column, and then calculate another column that is the total contribution margin. So for the contribution margin per unit, what we would do is we would subtract the cost from the price. And then for the total contribution margin, we'd take that new calculated contribution margin per unit and multiply it by the quantity.

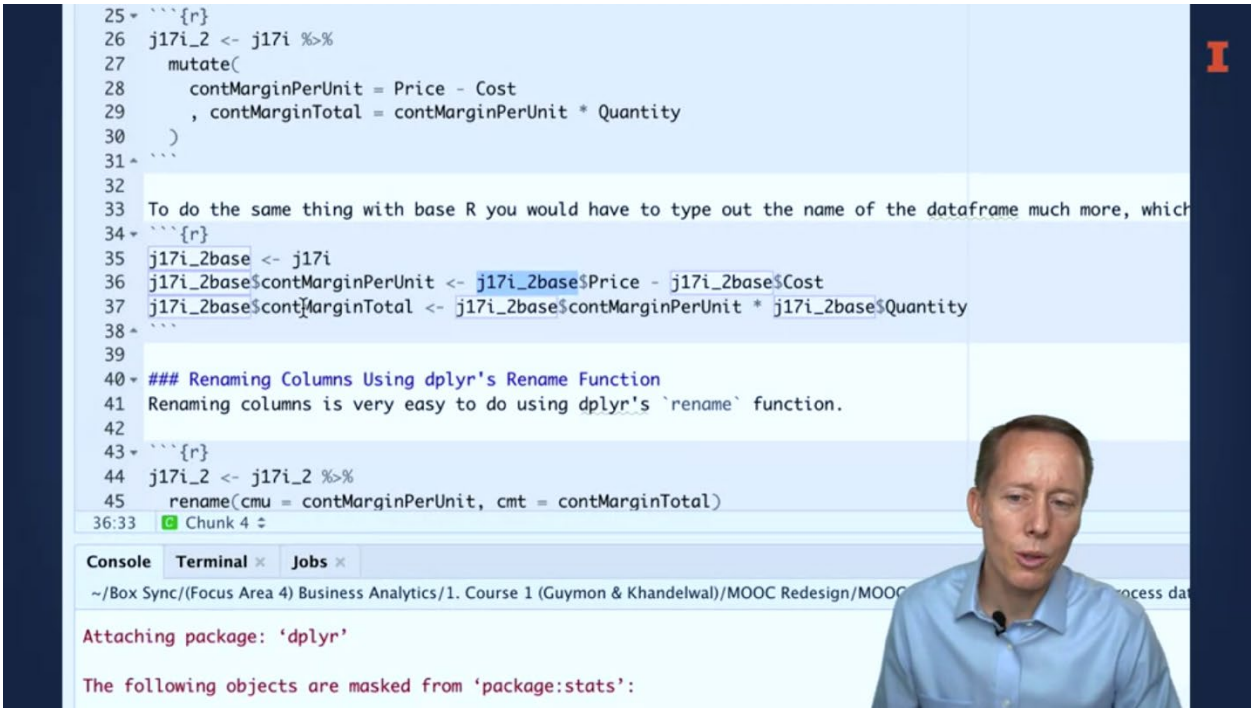


So it's pretty straightforward to do that using the mutate function. I'm going to create a new data frame object, j17i_2. And that will be the result of taking the j17i data frame object and piping it into the mutate function. Now, the way the mutate function works is, you type out the name of the new column that you want to calculate and set it equal to the calculation. So the contMarginPerUnit column will be equal to the price minus the cost, and then the contMarginTotal column will be equal to this new contMarginPerUnit column that we just created times the quantity. So you can see, you don't have to wait until you've run this to then use that new column in another calculated column. You can do it all in here, but it does have to be in order. Now I have this comma at the beginning of the second line here. Why do I do that? I can see how it makes more sense to read if you put it the comma at the end of every row there. The reason why I don't do that is because sometimes I have columns that I calculate that I don't want to get rid of. I don't want to totally delete, I want to save for later, and so I may want to comment them out. And if I have that comma at the beginning of the row, I don't have to delete a comma and then comment out the row. So that's just a little pro tip. You can take it or leave it.



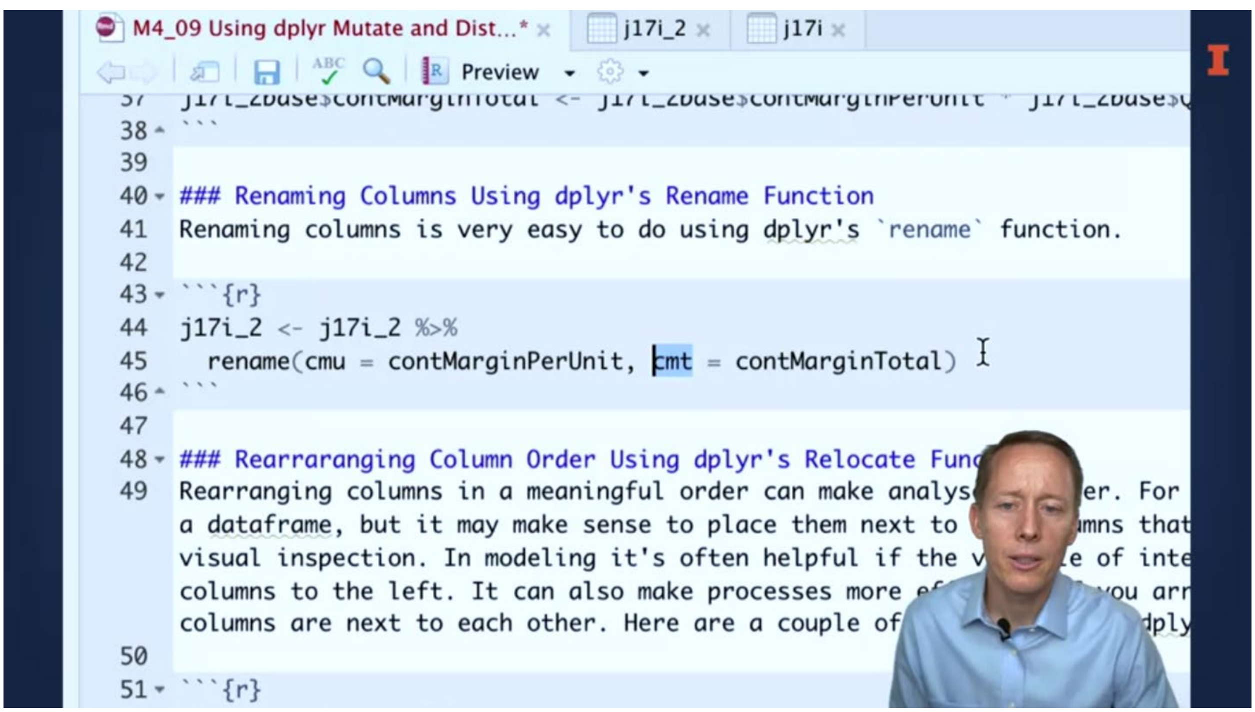
Cost	Price	Quantity	contMarginPerUnit	contMarginTotal
0.11	3.73	1	3.62	3.62
0.11	13.38	1	13.27	13.27
0.11	13.90	1	13.79	13.79
0.11	2.88	1	2.77	2.77
0.11	4.83	1	4.72	4.72
0.11	12.02	1	11.91	11.91
0.11	12.02	1	11.91	11.91
0.11	14.68	1	14.57	14.57
0.11	12.02	1	11.91	
0.11	4.03	1	3.92	
0.11	NaN	1	NaN	
0.11	2.88	1	2.77	
0.11	12.02	1	11.91	
0.11	14.68	1	14.57	

All right, let's go ahead and just run this code chunk, and let's look at the j17i_2 data frame object, and let's scroll to the far right. And sure enough, we've got the contMarginPerUnit and the contMarginTotal. And if we just visually inspect this, we can verify that it worked correctly. If we take 3.73 minus 0.11, yep, that's 3.62. And if we multiply that by 1, it's 3 62. Let's look at an observation where the quantity is not 1, to see if it work. Now I can't do this in my head, but 0.75 minus 0.13 is 0.62, and then multiplied by 36. That seems to be right, 22.32.



All right, it's always helpful to just make sure the calculations did what you think they were supposed to do. All right, so that's the mutate function. It's very helpful. For a comparison sake, if you were to do the same thing using base r, here's what you would have to do. You'd have to, well, first, I'll create a separate data frame object, j17i_2base, and I will just give that, the j17i data frame object. So it's just a copy of it. And then for the contMarginPerUnit column, I have to first type out the data frame name and then the dollar sign, and then set that equal to the price minus the cost. But you can see, I'm typing out the data frame name many times here, and then I would do a very similar thing for calculating the contMarginTotal.

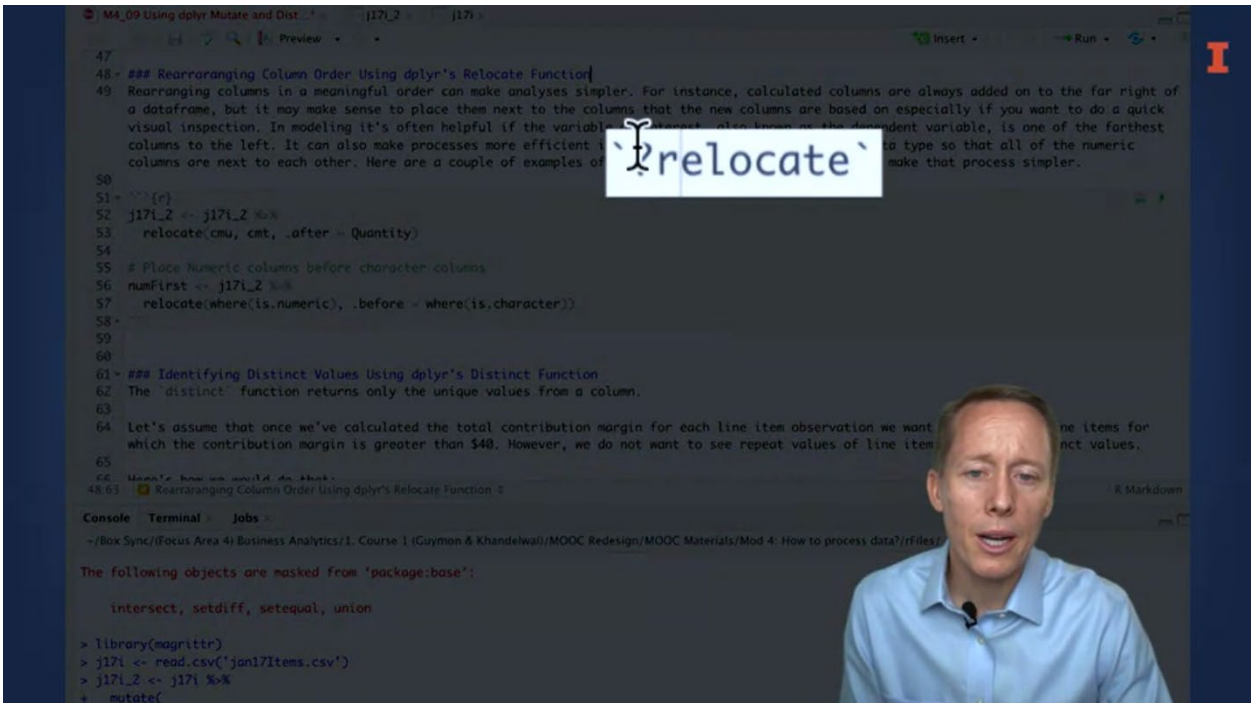
So if I run this code chunk, And look at this, I could quickly inspect it, and it would be the exact same thing as j17i_2. It's just that it's more verbose to type that out. All right, so that's the mutate function. It's a very, very useful function. Now let's talk about renaming columns using the rename function. This one is super simple.

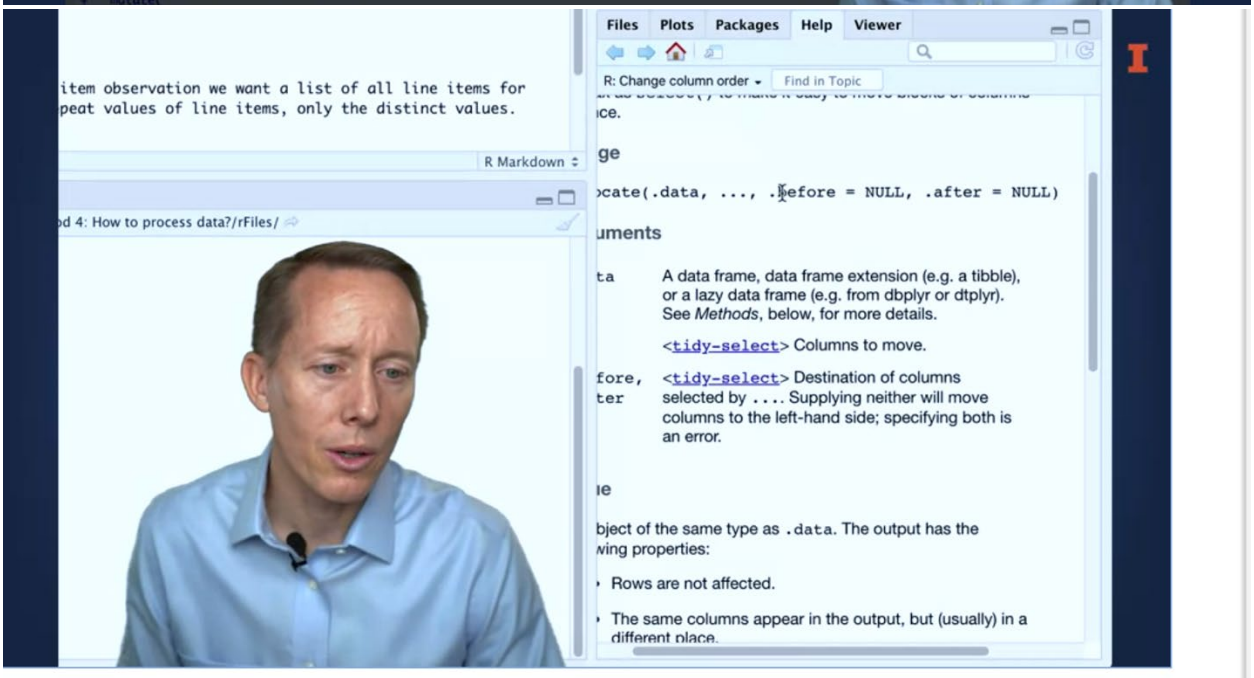


So let's assume that we want to shorten the name of the contMarginPerUnit column and the contMarginTotal column, they're too long. Using the rename function, I can take the

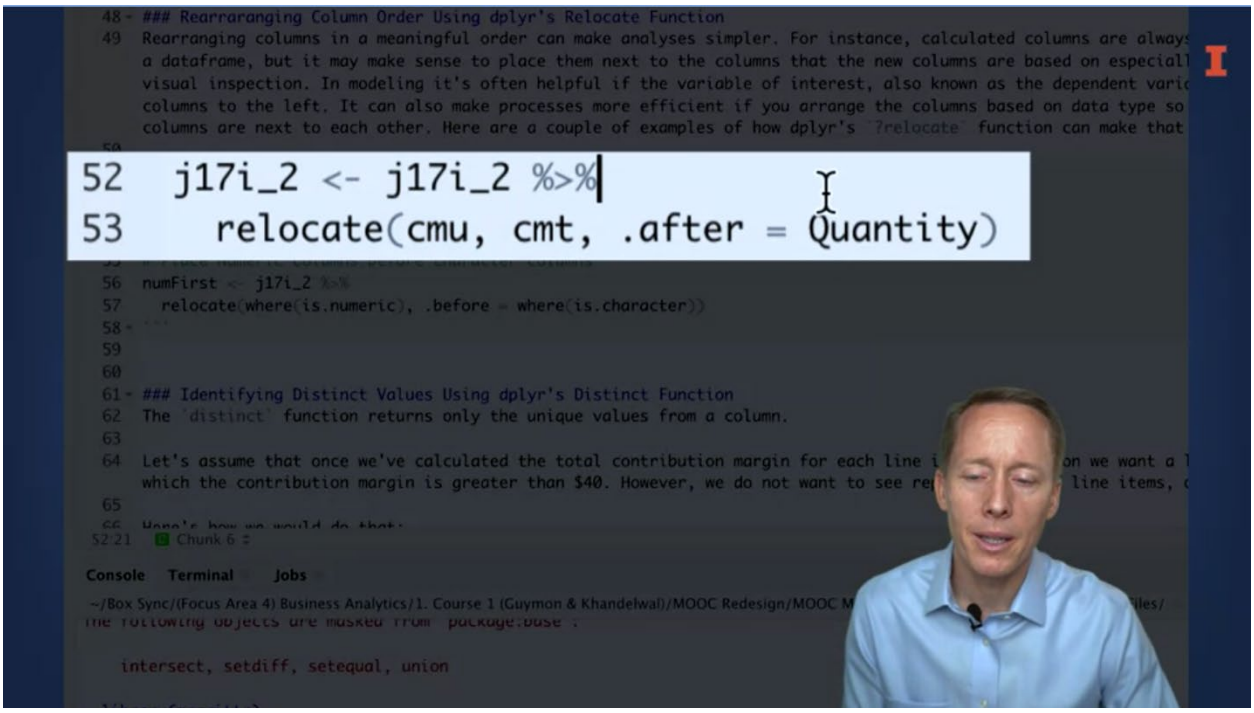
data frame object, pipe that into the rename function, and I simply type out the new name for a column and set that equal to the existing column name. All right, so you can see I'm renaming the contMarginPerUnit column to cmu and the contMarginTotal column to cmt.

So if I run this code chunk and look at j17i_2, scroll to the end, it now has the new name, cmu and cmt. All right, super simple. Now let's talk about rearranging the order of columns. This is an important pre-processing task. In our case, we might want to put the contribution margin per unit, contribution margin total columns right next to the quantity, price, and cost columns. For modelling purposes, oftentimes you want to put the key variable called the dependent variable on the far left of the data frame. And sometimes it just makes sense to put numeric columns right next to each other, and then character columns next to each other, and date columns next to each other. So, rearranging columns is a very common pre-processing task.

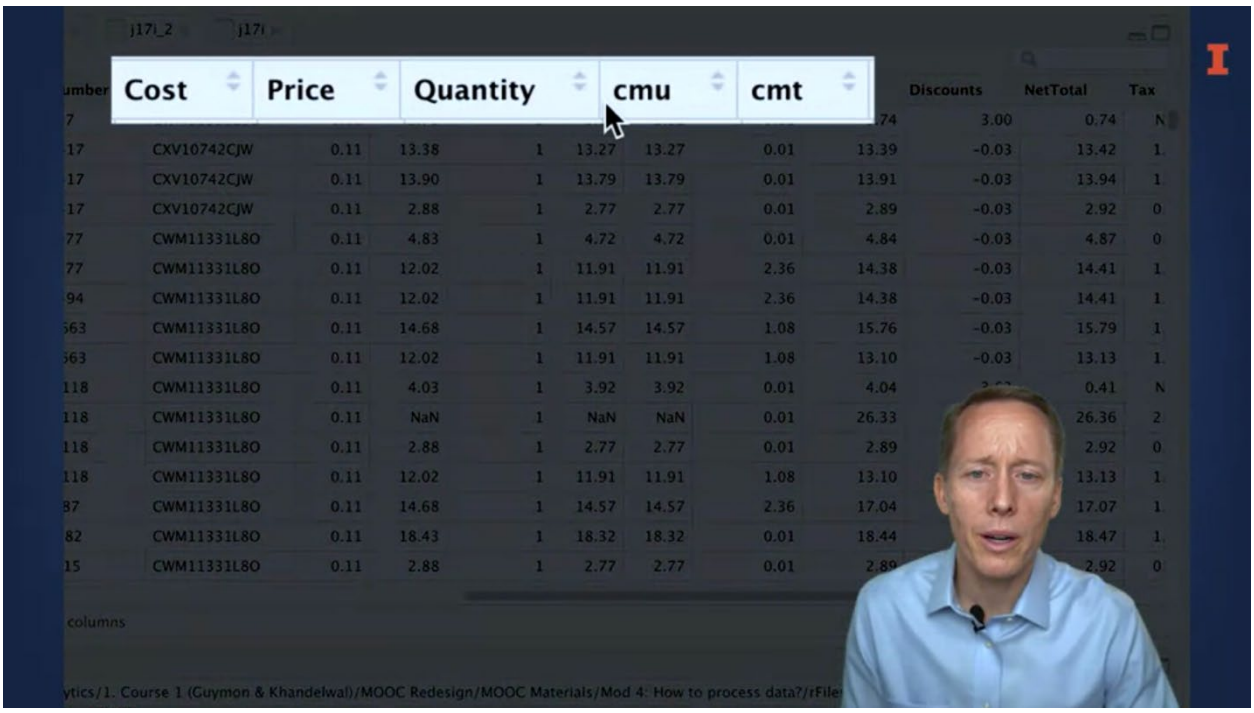




So let's read about how to use the relocate function in the dplyr package by reading the documentation for that. It's pretty easy to use. You can see that the first argument is the data, and after that there's the ellipses and that just refers to the columns that you want to move, and then there are options that you can use for indicating where they should go. So you can use .before or .after to place these columns that you want to move before or after a certain column. And then the examples at the bottom are very helpful. So I would encourage you to read those at some point.

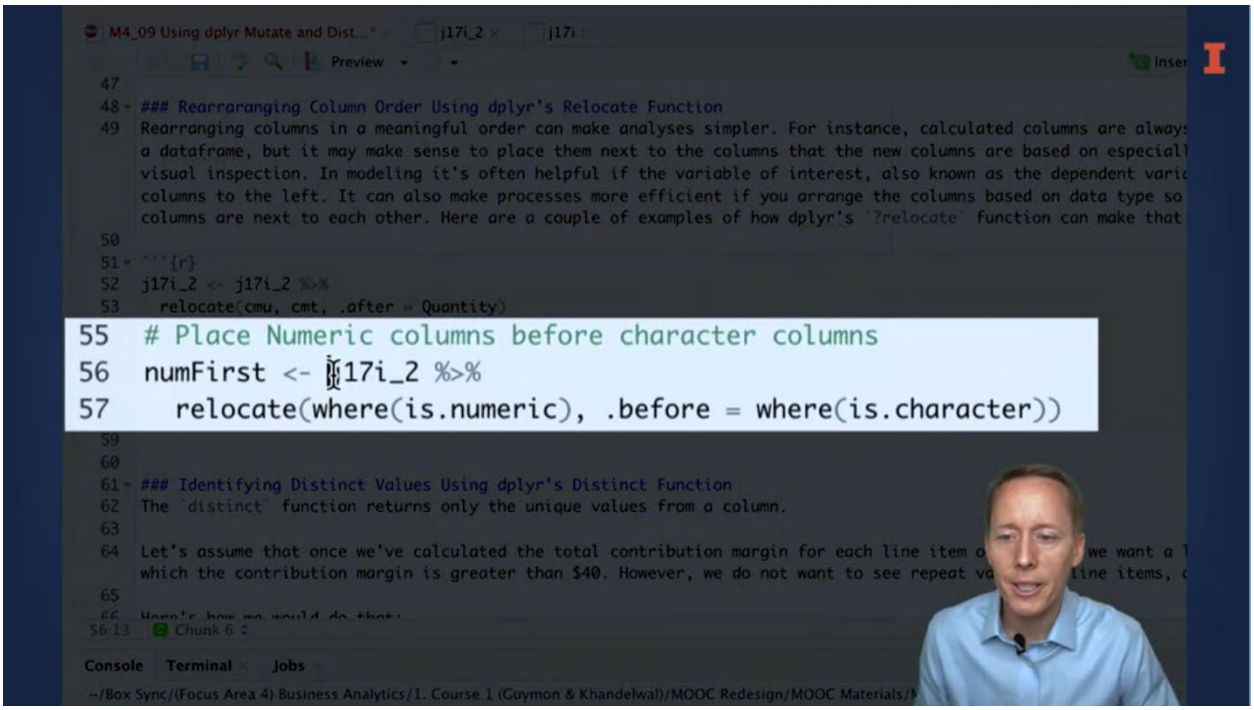


I'm just going to illustrate two of these examples here. So, the first example I want to provide is to move the contribution margin columns that we just created next to the cost, price, and quantity columns. So I'll start with the j17i_2 data frame object, pipe that into the relocate function, and then I will indicate by name, cmu and cmt. Those are the columns I want to move, and I want to put them after the quantity column.

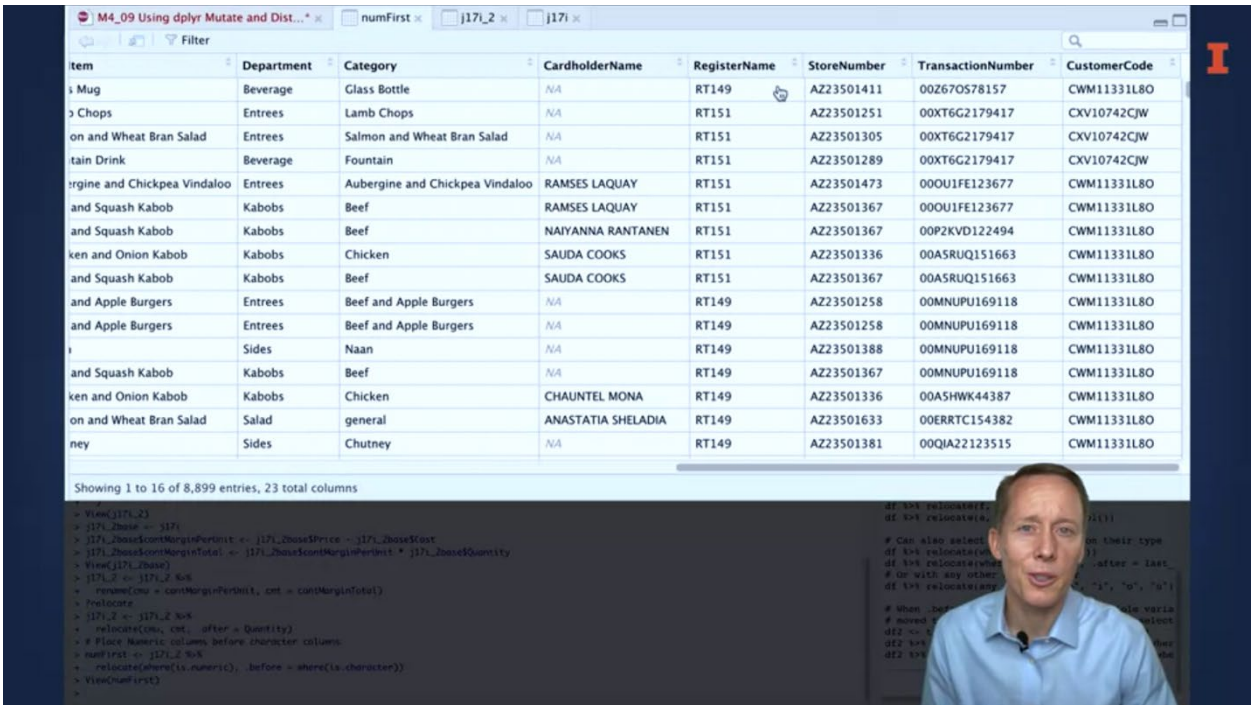


So let's go ahead and run that, and look at j17i_2, and here's the cost, price, quantity,

and then the cmu and cmt. So it worked, all right? It's very easy to read. Simple to use. Now, if you have a wide data frame object, rearranging column order is even more important. So let's say we want to put all the numeric columns as the first columns on the far left of the data frame object, followed by the character string columns.

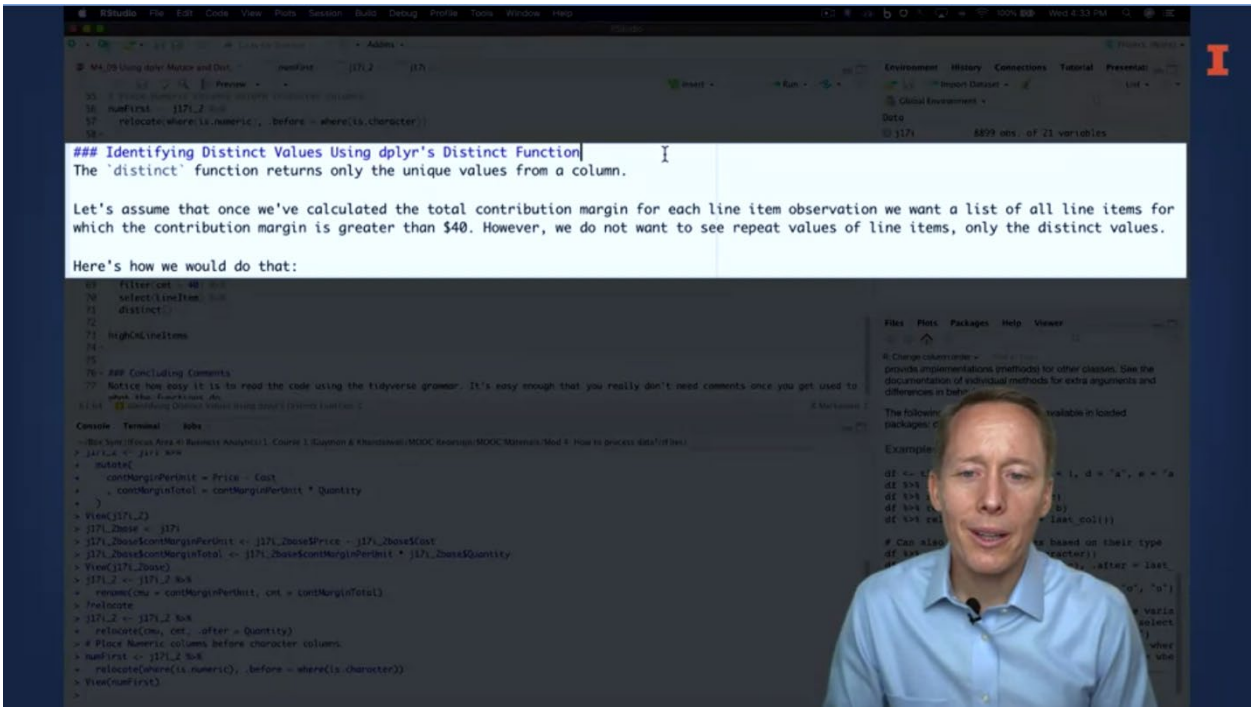


Here's how we would do that. We'll take the j17i_2 data frame object, pipe it into the relocate function, and then we'll use this where function and we're going to say, where the column data type is numeric? We're going to move those, and we'll put them before wherever there are character string columns, okay? So we use that where function, again, and we'll say wherever its character. And I will put that into a new data frame object, numFirst.

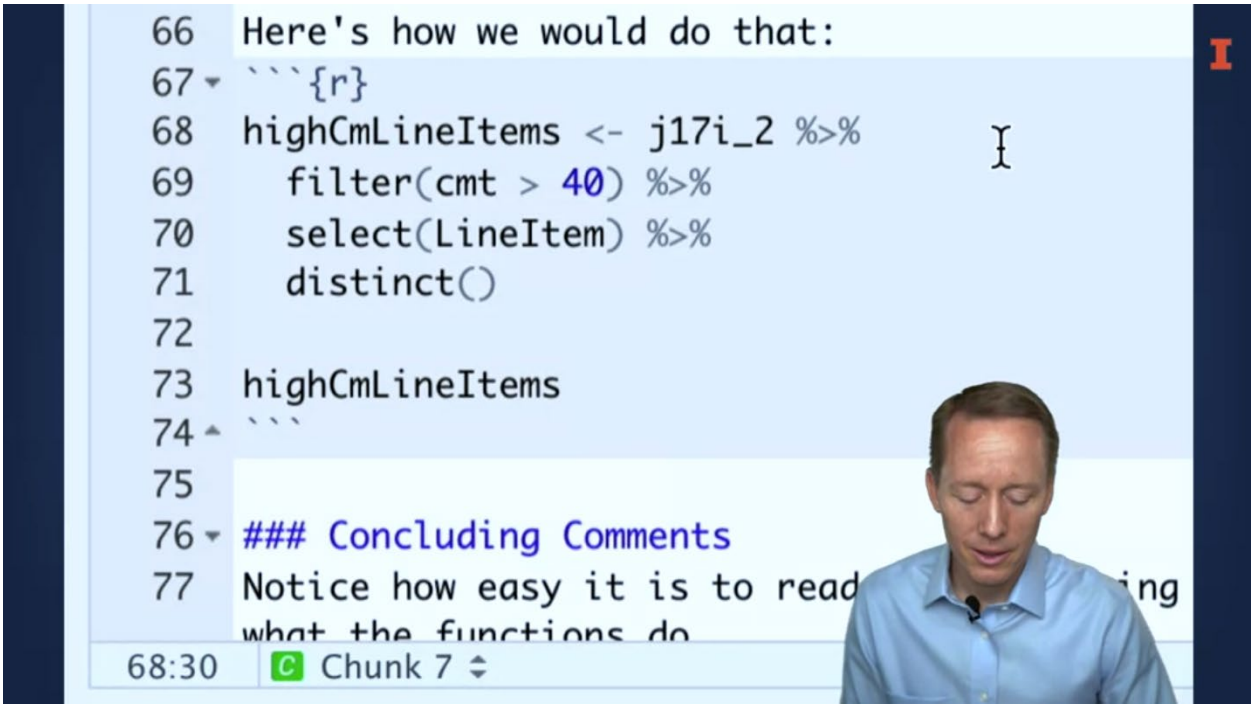


Item	Department	Category	CardholderName	RegisterName	StoreNumber	TransactionNumber	CustomerCode
s Mug	Beverage	Glass Bottle	NA	RT149	AZ23501411	00Z67OS78157	CWM11331L8O
o Chops	Entrees	Lamb Chops	NA	RT151	AZ23501251	00XT6G2179417	CXV10742CJW
on and Wheat Bran Salad	Entrees	Salmon and Wheat Bran Salad	NA	RT151	AZ23501305	00XT6G2179417	CXV10742CJW
tain Drink	Beverage	Fountain	NA	RT151	AZ23501289	00XT6G2179417	CXV10742CJW
rgine and Chickpea Vindaloo	Entrees	Aubergine and Chickpea Vindaloo	RAMSES LAQUAY	RT151	AZ23501473	00OU1FE123677	CWM11331L8O
and Squash Kabob	Kabobs	Beef	RAMSES LAQUAY	RT151	AZ23501367	00OU1FE123677	CWM11331L8O
and Squash Kabob	Kabobs	Beef	NAIYANNA RANTANEN	RT151	AZ23501367	00P2KVD122494	CWM11331L8O
ken and Onion Kabob	Kabobs	Chicken	SAUDA COOKS	RT151	AZ23501336	00A5RUQ151663	CWM11331L8O
and Squash Kabob	Kabobs	Beef	SAUDA COOKS	RT151	AZ23501367	00A5RUQ151663	CWM11331L8O
and Apple Burgers	Entrees	Beef and Apple Burgers	NA	RT149	AZ23501258	00MNUPU169118	CWM11331L8O
and Apple Burgers	Entrees	Beef and Apple Burgers	NA	RT149	AZ23501258	00MNUPU169118	CWM11331L8O
	Sides	Naan	NA	RT149	AZ23501388	00MNUPU169118	CWM11331L8O
and Squash Kabob	Kabobs	Beef	NA	RT149	AZ23501367	00MNUPU169118	CWM11331L8O
ken and Onion Kabob	Kabobs	Chicken	CHAUNTEL MONA	RT149	AZ23501336	00A5HWK44387	CWM11331L8O
on and Wheat Bran Salad	Salad	general	ANASTATIA SHELADIA	RT149	AZ23501633	00ERRTC154382	CWM11331L8O
ney	Sides	Chutney	NA	RT149	AZ23501381	00QIA22123515	CWM11331L8O

So let's go ahead and just make sure that happened. I'll just visually explore this. We go to the numFirst data frame object. You can see now, the first column on the far left is cost, price, quantity. All these are numeric, and then we get to time, time, operation type, barcode, and so forth. All the character string columns. All right, so that's how you can easily rearrange the order of columns. Now let's talk about how we can return only distinct values from a certain column.

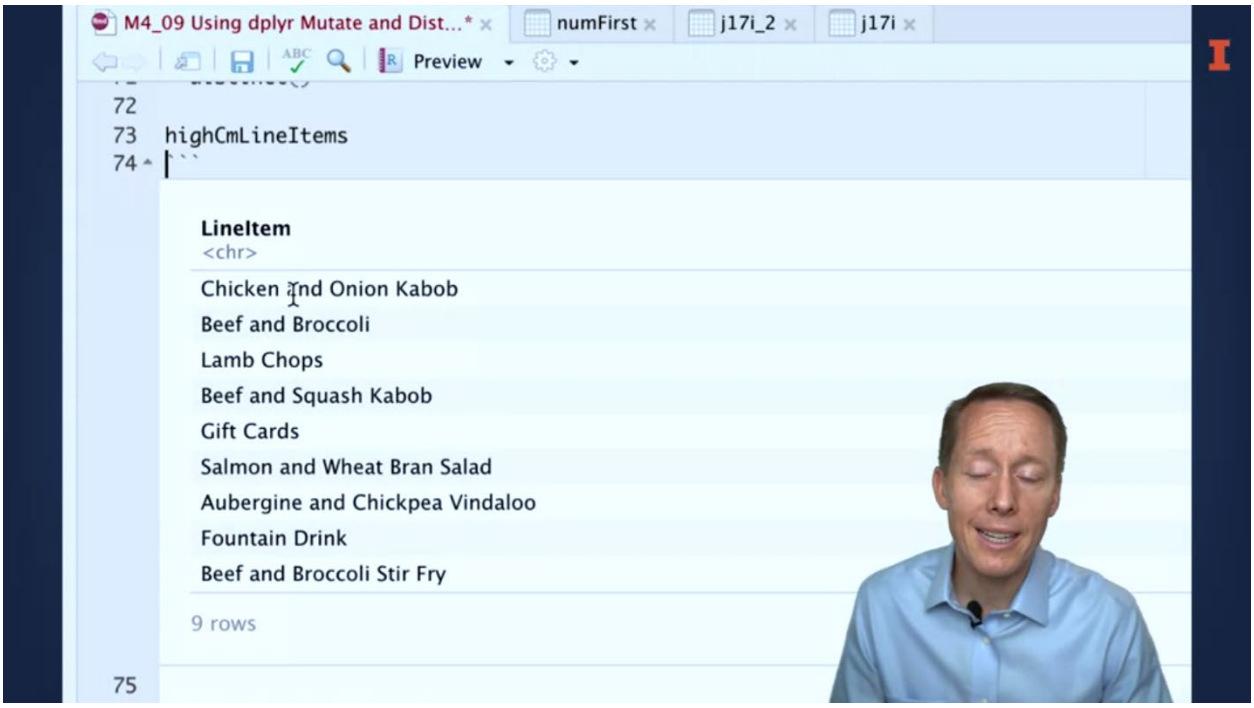


One way that we can do that is using the distinct function in the dplyr package. Let's assume that once we've calculated the total contribution margin for each line item observation, we want a list of all line items for which the contribution margin is greater than 40. However, we don't want to see repeated values of them. We only want to see the distinct values. Here's an example of how we can use the distinct function.



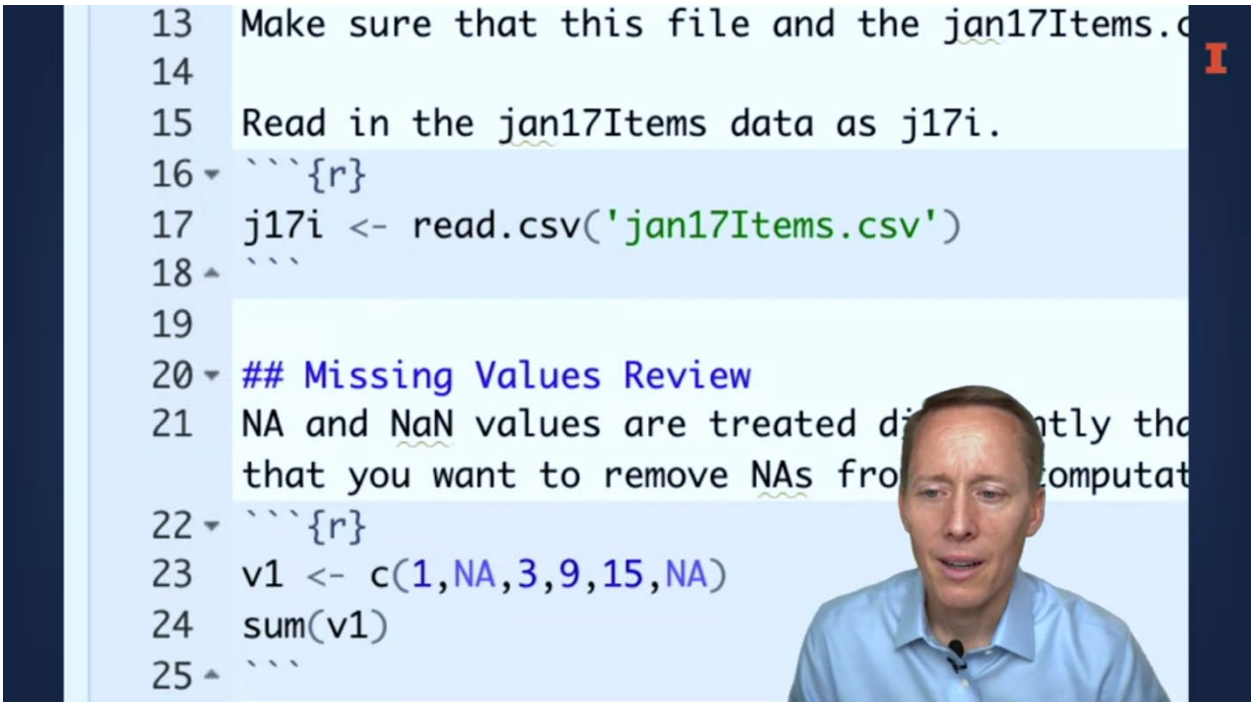
Let's take the j17i_2 data frame object, pipe that into the filter function, and we're only

going to keep observations for which the contribution margin total is greater than 40. Then we'll pipe that filter data frame object into the select function, and we're going to select only the line item column, and then we'll pipe that into this distinct function so that we're left with only the unique values.

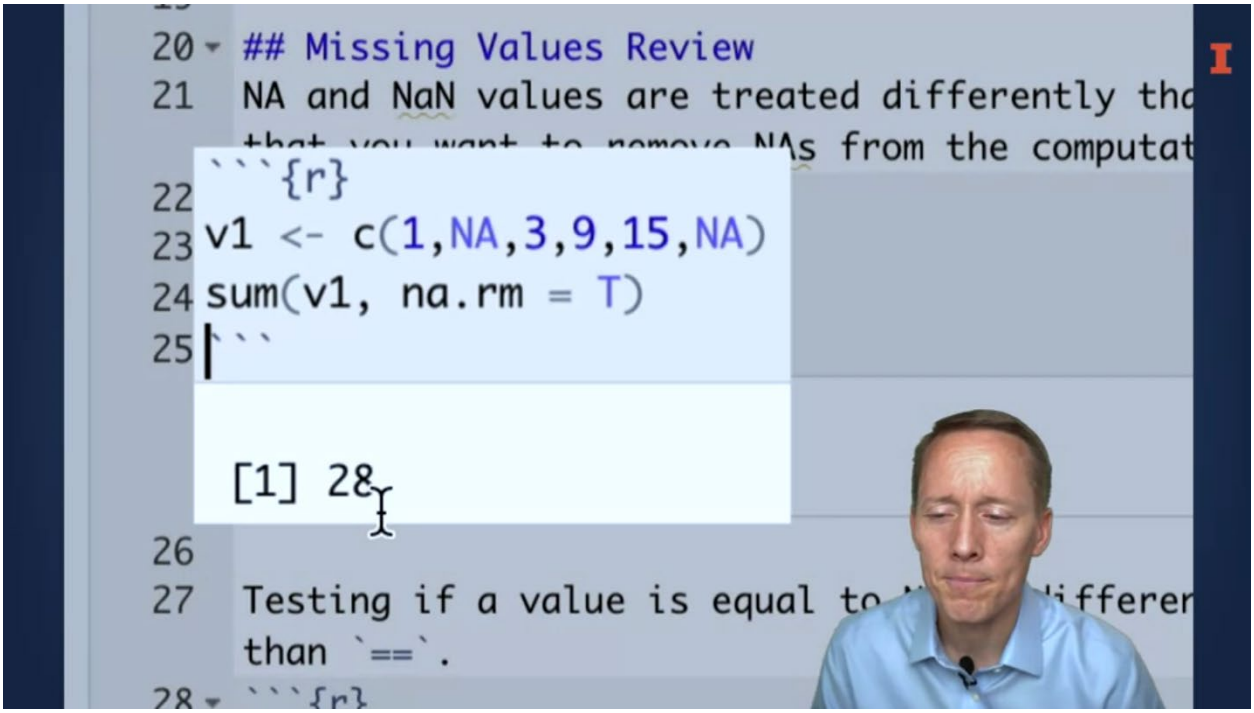


So I will run that, and then let's just print out those values. And sure enough, there's only nine rows in this new object here. And there, the distinct values for which the contribution margin total is greater than 40. In conclusion, notice how easy it is to perform all of these pre-processing tasks using the tidyverse grammar. It's simple enough that you really don't even need to use comments.

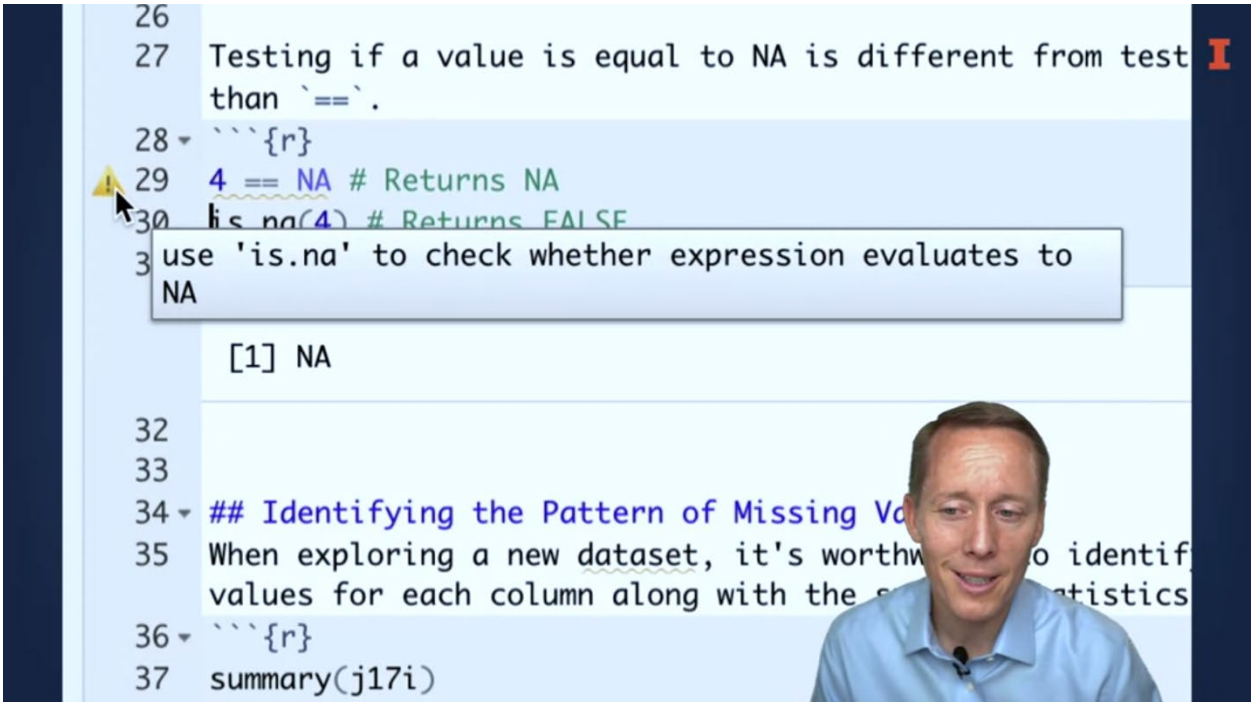
Lesson 4-1.10 Handling Missing Values



In this lesson, we're going to talk about how to deal with missing values, which are an inevitable part of business analytics. So a couple of approaches that you can take for dealing with missing values is to either remove the columns or the observations or to impute the missing values. So let's demonstrate how to do this. First of all, make sure that you have installed and loaded the dplyr and magrittr packages. And then make sure that you have the jan17Items.csv file that you can read in. All right, now, let's just spend a minute and review missing values. Missing values are represented by NA for not applicable or NAN, which stands for not a number, and they're treated differently than numeric values.

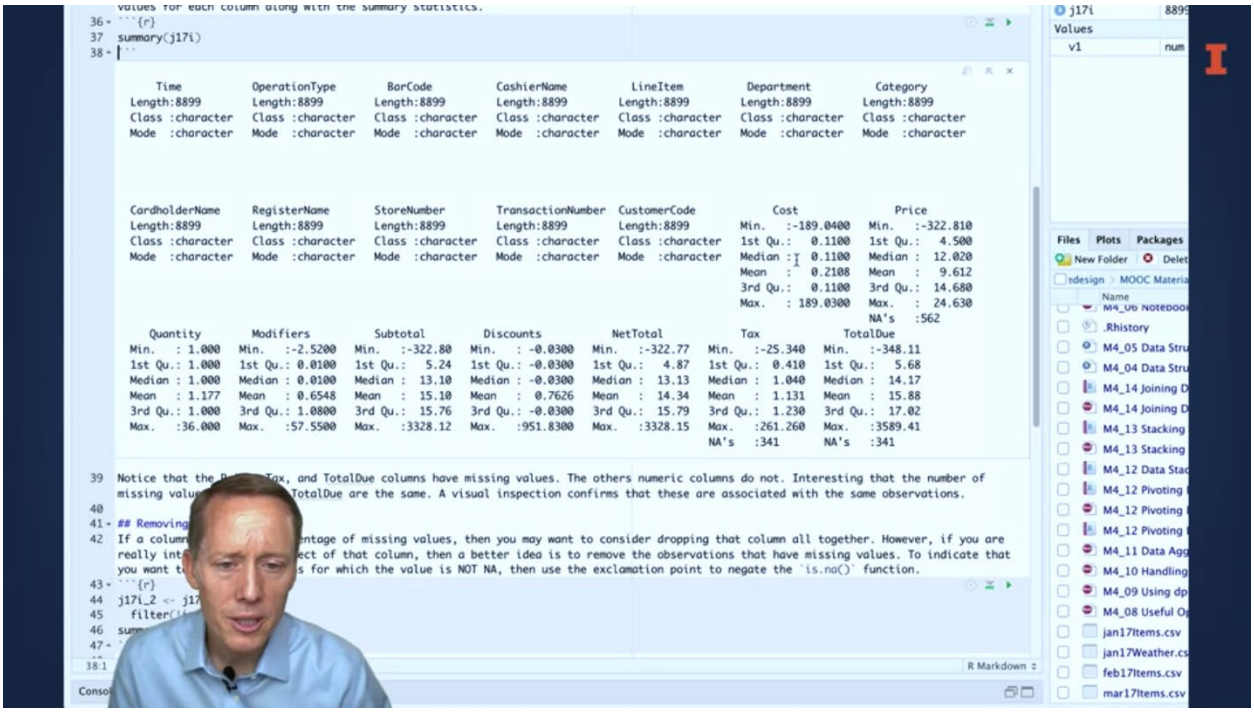


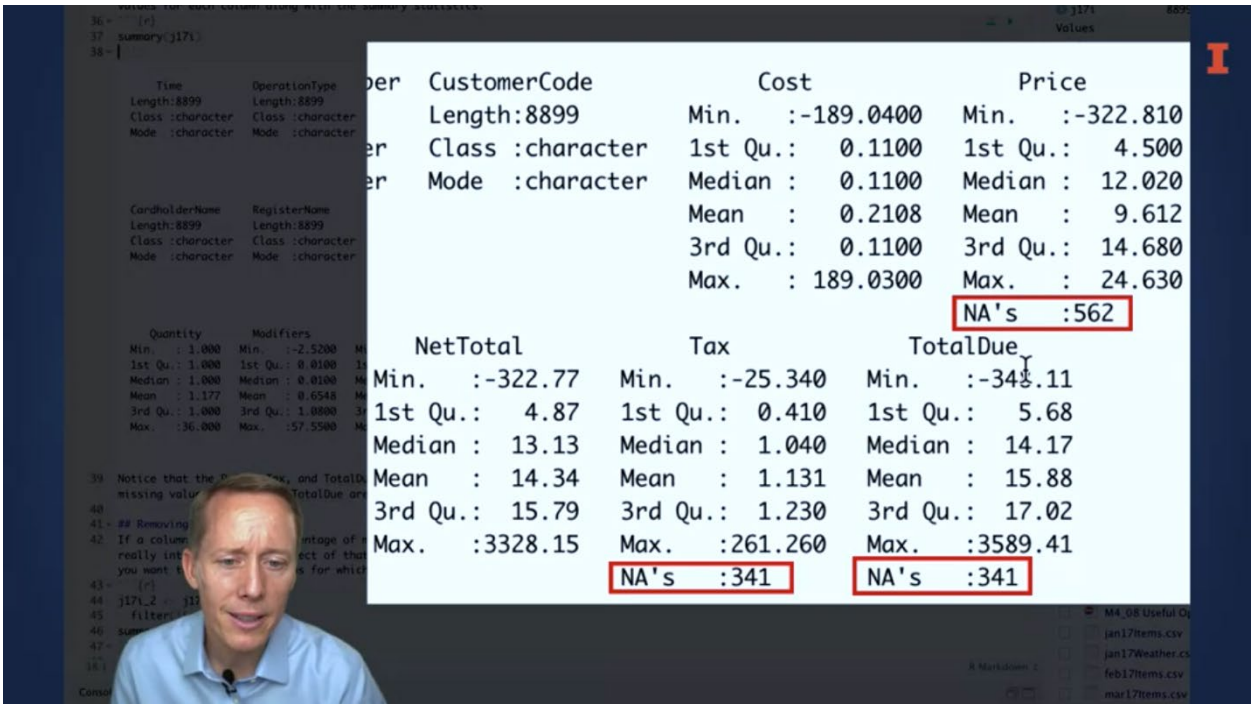
For instance, if we have a vector in which we have about seven different observations and some of those are NA. And we try to sum the values in that vector, we'll get NA. And so we have to explicitly identify that we want to remove the missing values. And when we do that, it'll work out.



Also, if we want to test if missing values are present, it's different than testing if a substring is present or if the value is equal to some other number. So the way we do

that is by using the is.na function. So you can see here I'm testing if 4 is equal to NA. If I run that, it will return NA and looking at this exclamation point here this warning our studios helpful, it says, hey, I think I know what you're trying to do. And the way to do that correctly is to use the is.na function to check whether it is NA. So I'll try that on this row here, and it says false. because 4 is not NA, it's not a missing value. All right, now, when we analyze business data, we often want to look at the pattern of missing values.

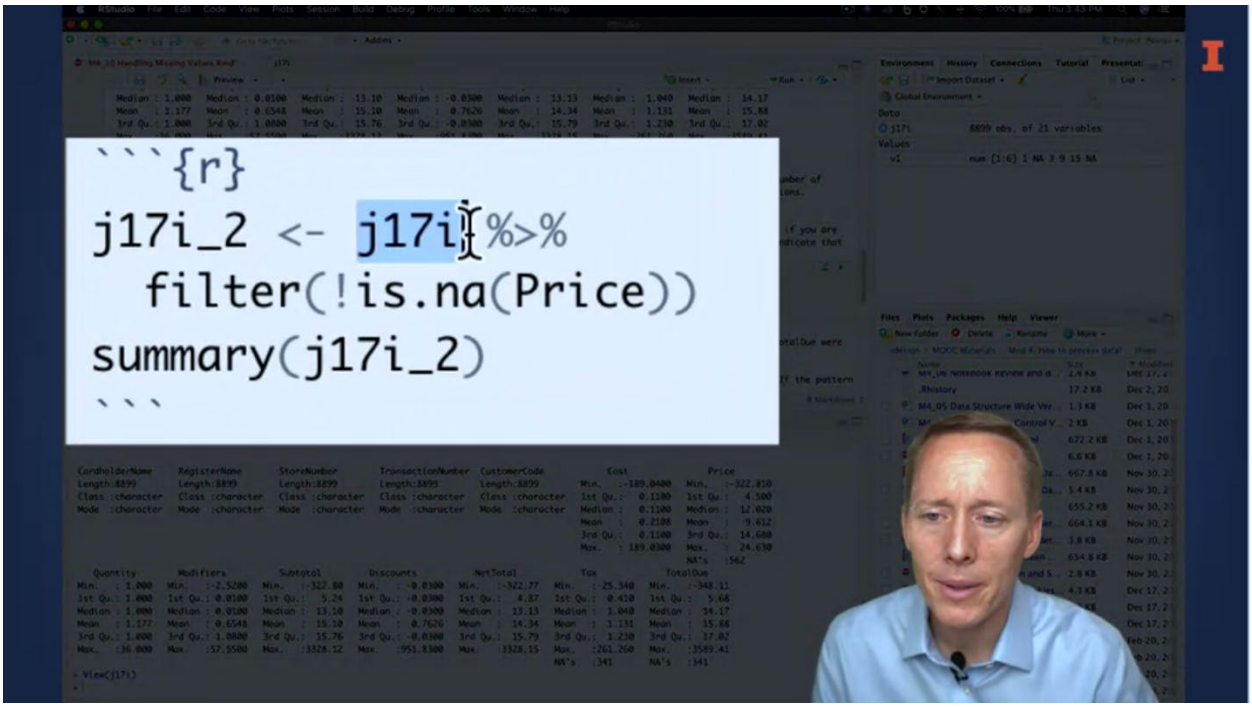




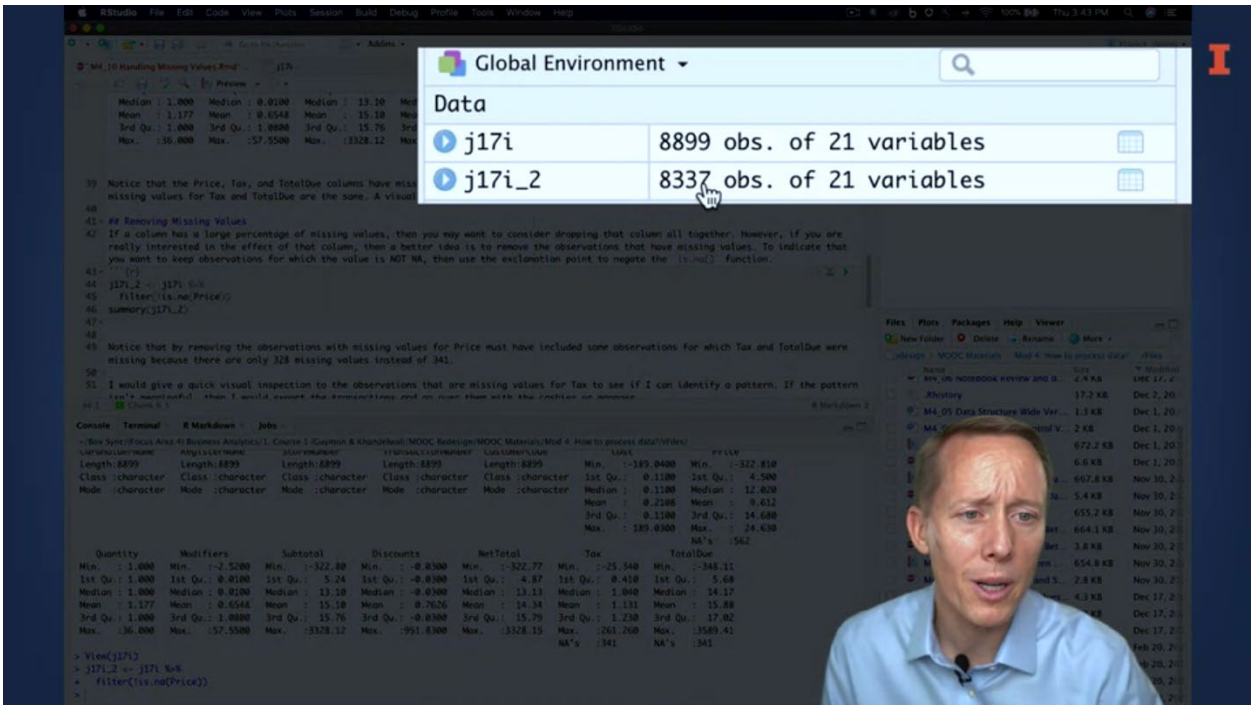
And one way that we can do that is by using the summary function. And I'll run that on this j17i dataframe. And as you look at these different columns here, especially the numeric columns, we can see that price has 562 missing observations. And tax and TotalDue both have 341 missing observations. Interesting that the number of observations is the same for tax and TotalDue.

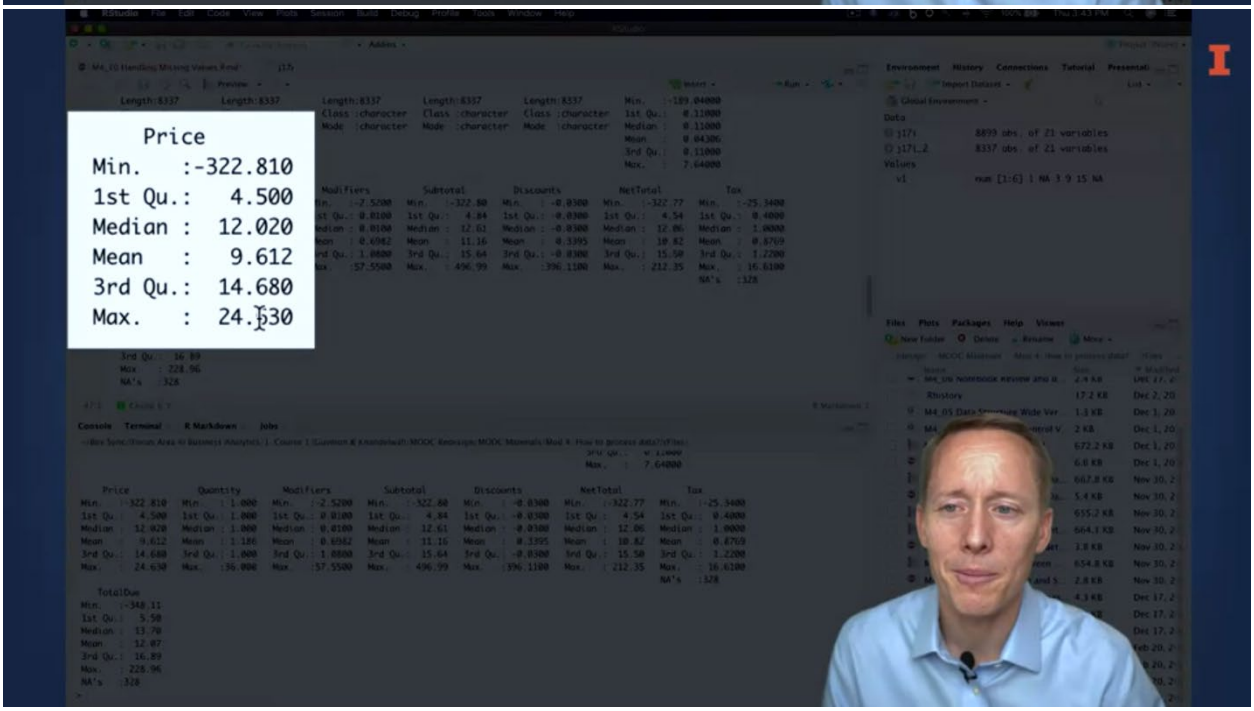
ime	StoreNumber	TransactionNumber	CustomerCode	Cost	Price	Quantity	Modifiers	Subtotal	Discounts	NetTotal	Tax	TotalDue
	AZ23501258	00T99K472070	CWM11331L8O	0.11	NaN	1	0.01	30.60	6.17	24.43	1.92	26.35
	AZ23501305	00T99K472070	CWM11331L8O	0.11	NaN	1	0.01	29.52	5.85	23.67	1.86	25.53
	AZ23501350	00J10KA62215	CWM11331L8O	0.11	13.38	1	1.08	14.46	-0.03	14.49	1.13	15.62
	AZ23501404	00FQW2B174660	CWM11331L8O	0.11	2.88	1	0.01	2.89	-0.03	2.92	0.23	3.15
	AZ23501258	00CGMV4154080	CWM11331L8O	0.11	2.27	1	0.01	2.28	1.46	0.82	NaN	NaN
	AZ23501258	00CGMV4154080	CWM11331L8O	0.11	11.52	1	0.01	11.53	-0.03	11.56	0.90	12.46
	AZ23501305	00CGMV4154080	CWM11331L8O	0.11	13.13	1	0.01	13.14	-0.03	13.17	1.04	14.21
	AZ23501411	00CGMV4154080	CWM11331L8O	0.11	3.73	1	0.01	3.74	-0.03	3.77	0.30	4.07
	AZ23501367	00CGMV4154080	CWM11331L8O	0.11	12.02	1	1.08	13.10	-0.03	13.13	1.04	14.17
	AZ23501305	00K8Y0U172818	CWM11331L8O	0.11	17.43	1	0.01	17.44	-0.03	17.47	1.37	18.84
	AZ23501305	00K8Y0U172818	CWM11331L8O	0.11	8.71	1	0.01	8.72	-0.03	8.75	0.69	9.44
	AZ23501633	00K8Y0U172818	CWM11331L8O	0.11	18.43	1	0.60	19.03	-0.03	19.06	1.50	20.56
	AZ23501305	00K8Y0U172818	CWM11331L8O	0.11	14.72	1	0.01	14.73	-0.03	14.76	1.16	15.92
	AZ23501596	00K8Y0U172818	CWM11331L8O	0.11	4.50	1	0.01	4.51	-0.03	4.54	0.35	4.89
	AZ23501305	00C4WNS114516	CWM11331L8O	0.11	NaN	1	0.01	29.10	-0.03	29.13	2.29	31.42
	AZ23501411	00C4WNS114516	CWM11331L8O	0.11	3.73	1	0.01	3.74	-0.03	3.77	0.30	4.07

If I were to visually explore the dataframe and just look at tax and TotalDue, it looks like the ones at least a quick visual inspection here. The observations for which the values are missing are the same observation. All right, so that's always helpful to just do a quick visual exploration of the data like that. Now let's talk about what we do know when we do know that we have missing data in our dataframe, the first and easiest approach is to just remove the columns for which there are missing values. That's not a very worthwhile approach, especially if you're interested in understanding, for instance, the tax, total due, or price column. So the next approach is rather than remove the column remove the observations for which the values are missing. And here's how we can do that.

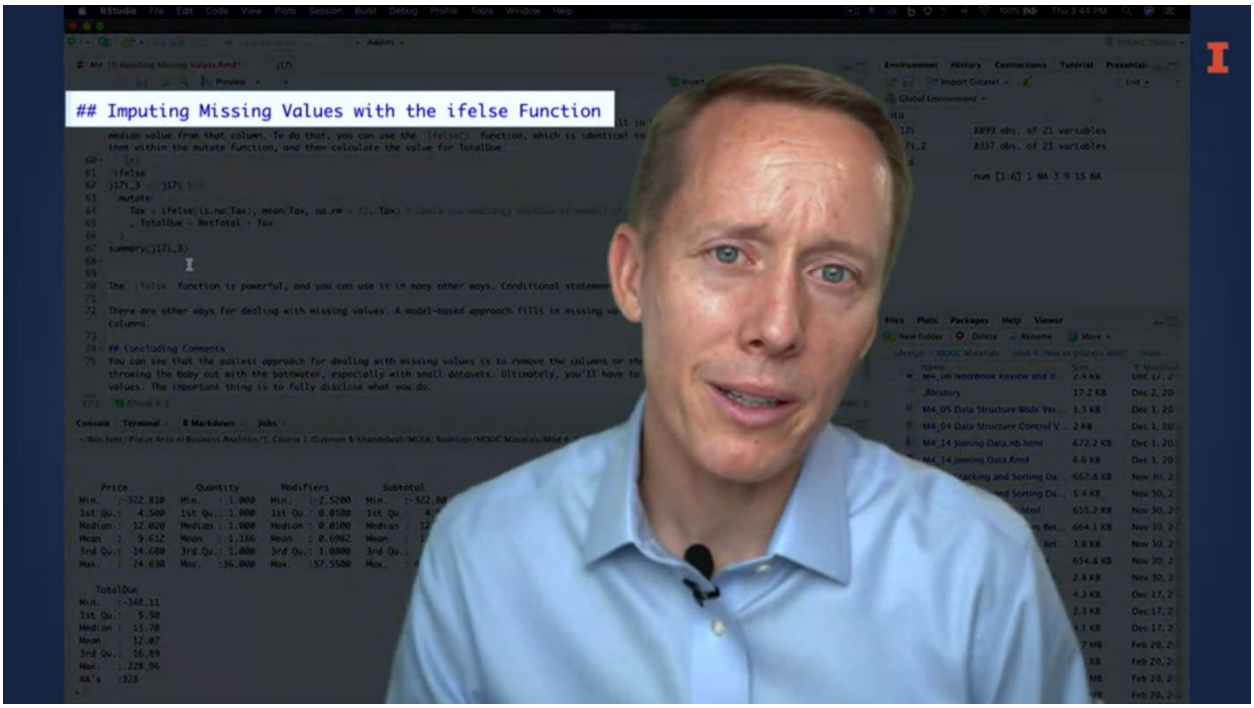


I will take the j17i dataframe and pipe it into the filter function, and rather than use the is.na(Price) alone, I need to negate that. And I do that by using the exclamation point, okay? So by putting that exclamation point in front of is.na it says, filter down to observations for which the price is not missing. So let's go ahead and try that out.

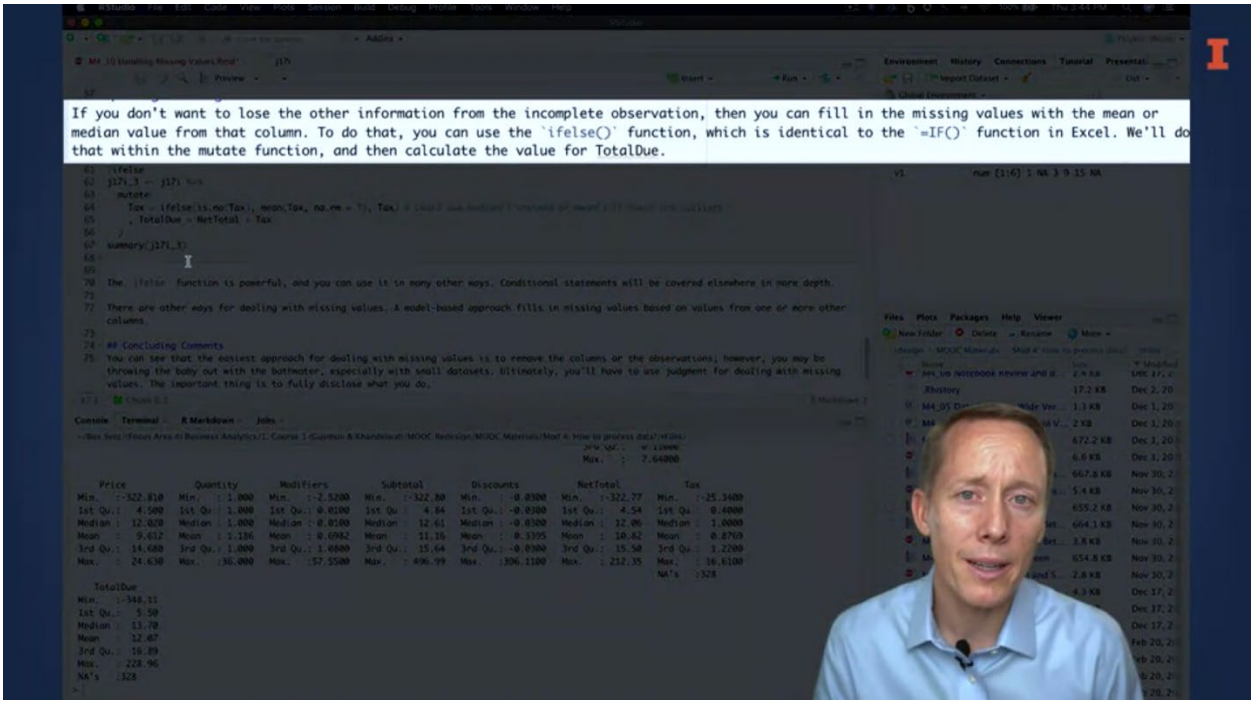




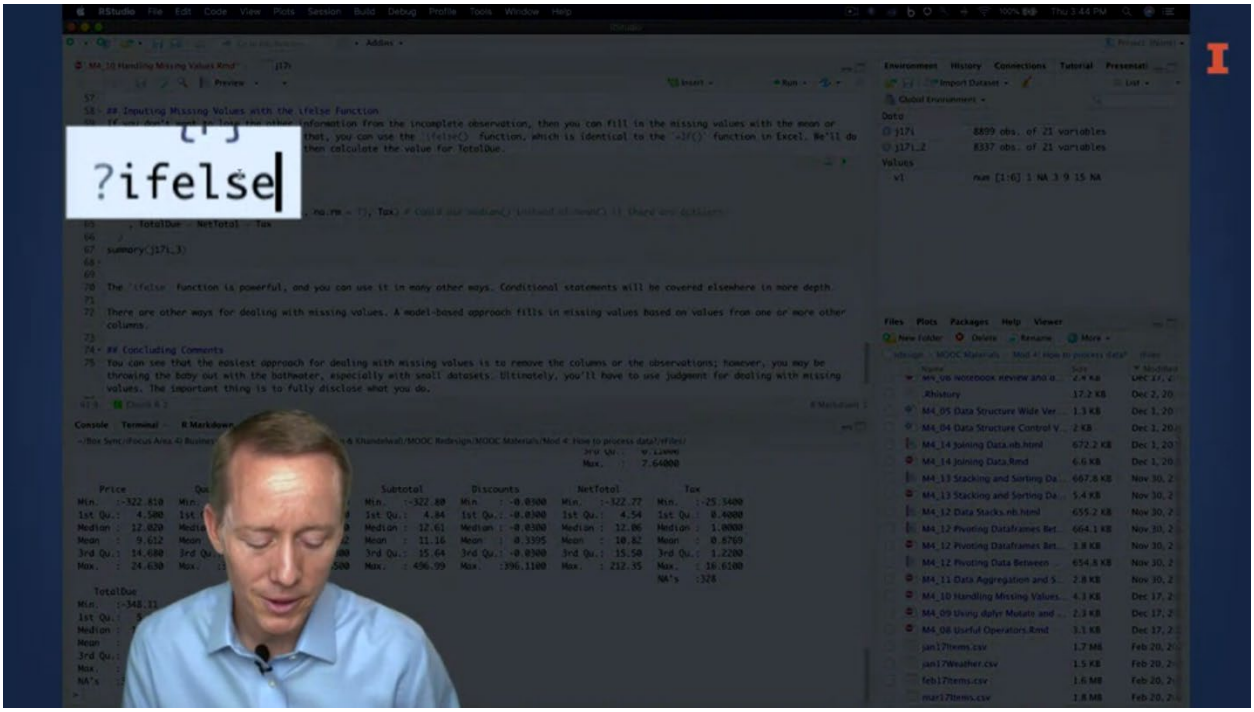
And we can see that we've got a fewer number of observations. And if I look at a summary on this new dataframe, I can go to price and see that sure enough, there are no NA's anymore. All right, so that's also a worthwhile approach. But also that could be a little bit too extreme, especially if we're dealing with a small dataframe.

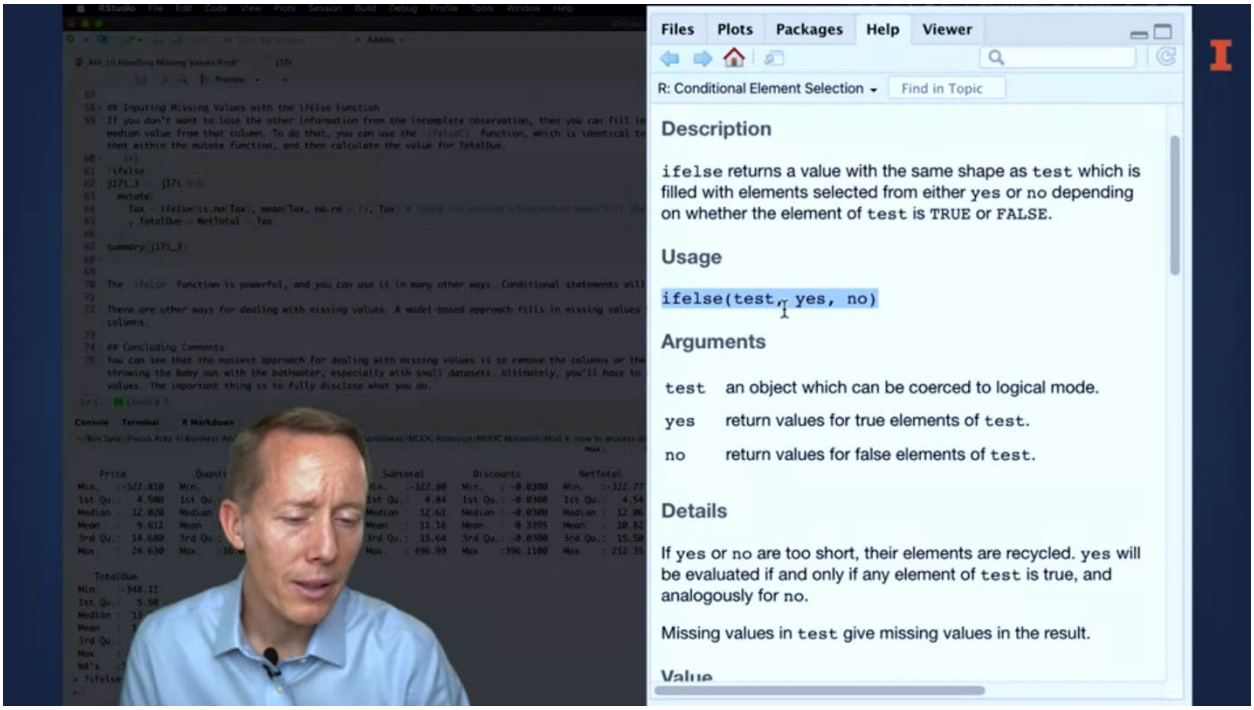


So another approach that we could take is to impute the missing values. Now I want to talk about what that means. Imputing missing value says, hey, let's take a guess at what that missing value could be. And there are some very sophisticated ways for imputing missing values.

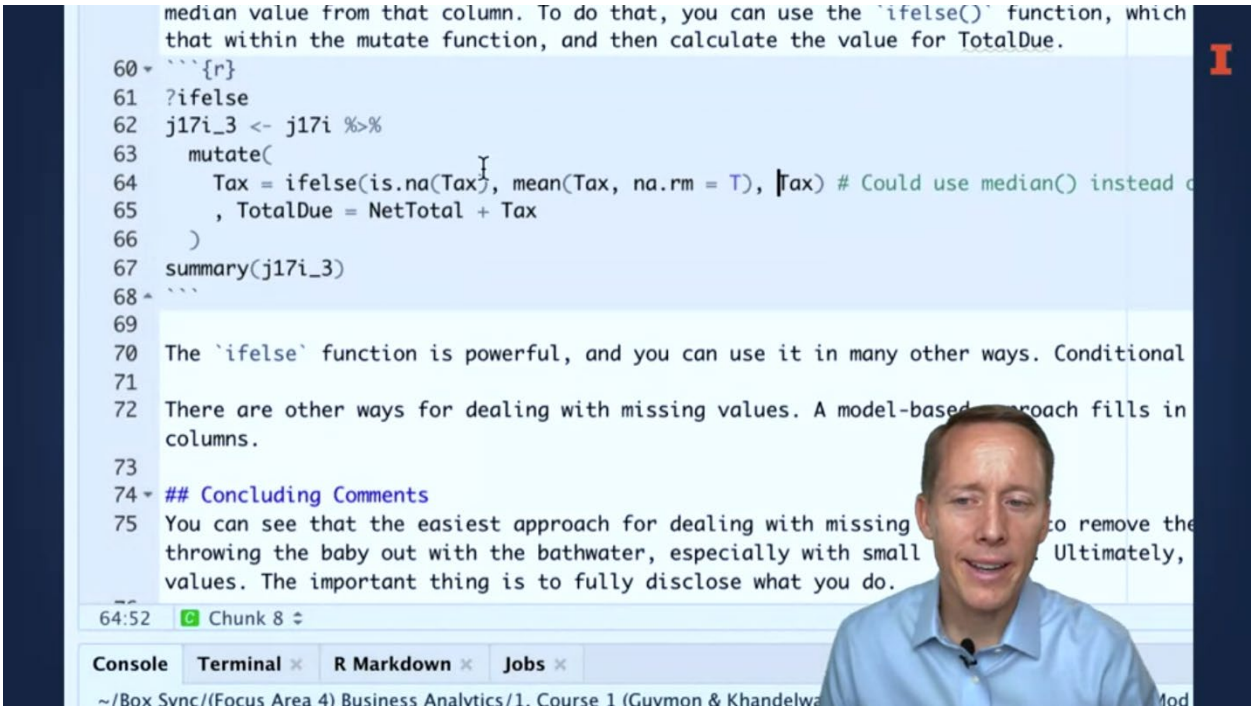


A very simple approach for doing that is to use either the mean or the median of that column.

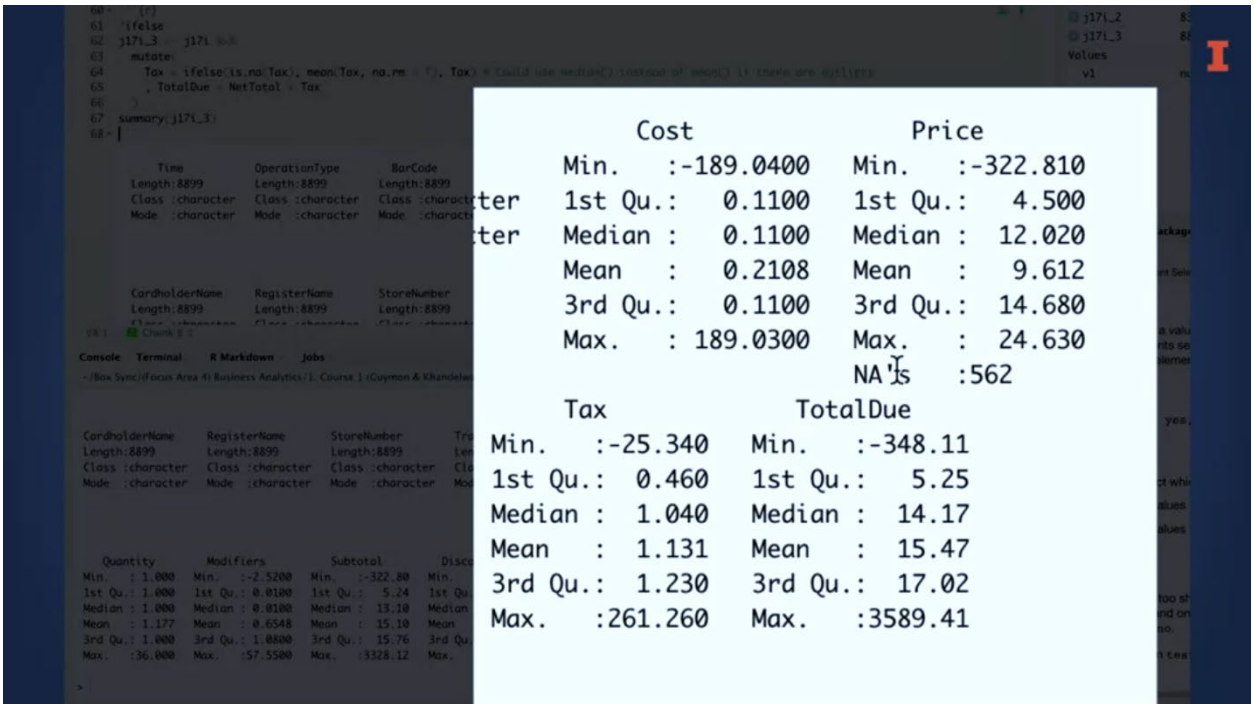




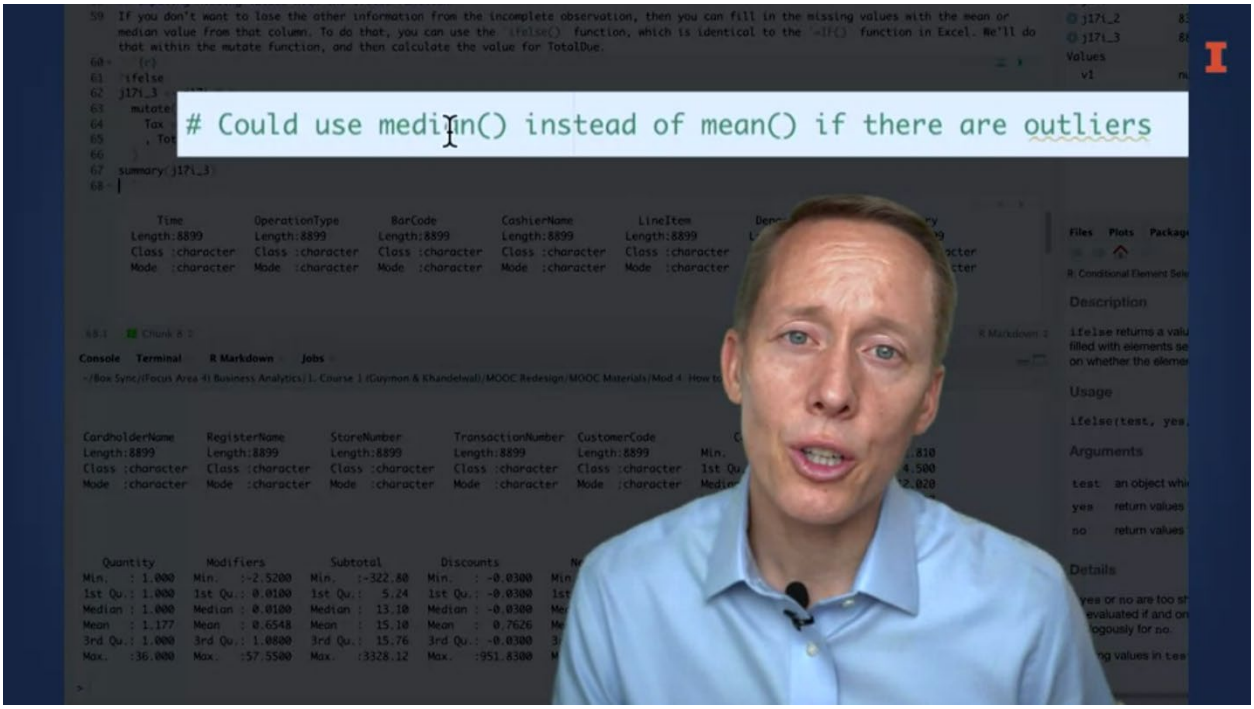
Now, to illustrate how this works in R, I want to introduce a new function, which is the if/else function. Let's read the documentation for this function. Essentially, the way this function works is that within the parentheses you perform a test. And when that test is true, you indicate what happens. When that test is false you indicate what the alternative should be. All right, so it's actually identical to how the if function is used in Excel.



So to use that in this j17i dataframe, to impute the missing values we'll take that data frame. So we'll create a new tax column and we'll use the if/else function. And the test we'll perform is we want to evaluate if the tax value is missing. So is it NA, is it missing? If it is, then we will fill in the missing value with the mean of that column, okay? And if it is not missing, we'll just leave it as the value that is already in that column. And now we can fill in the missing values in the TotalDue column by taking TotalDue and setting it equal to the NetTotal + the Tax.

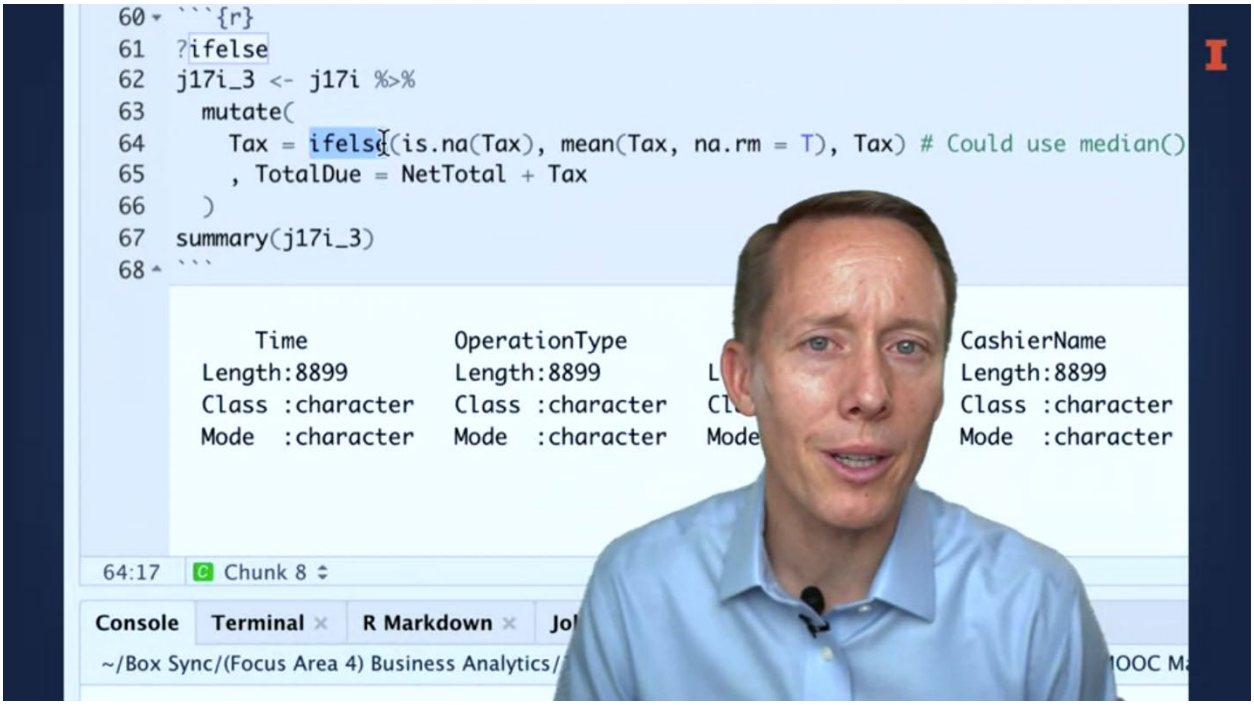


So let's go ahead and run that and look at a summary. And the price column still has the missing values because we saved that in a different dataframe object. But now tax and total due do not have any missing values. So it worked.



So we could replace missing values with the median. And we would especially want to do that if it is a skewed dataframe that has some very large observations. We may want

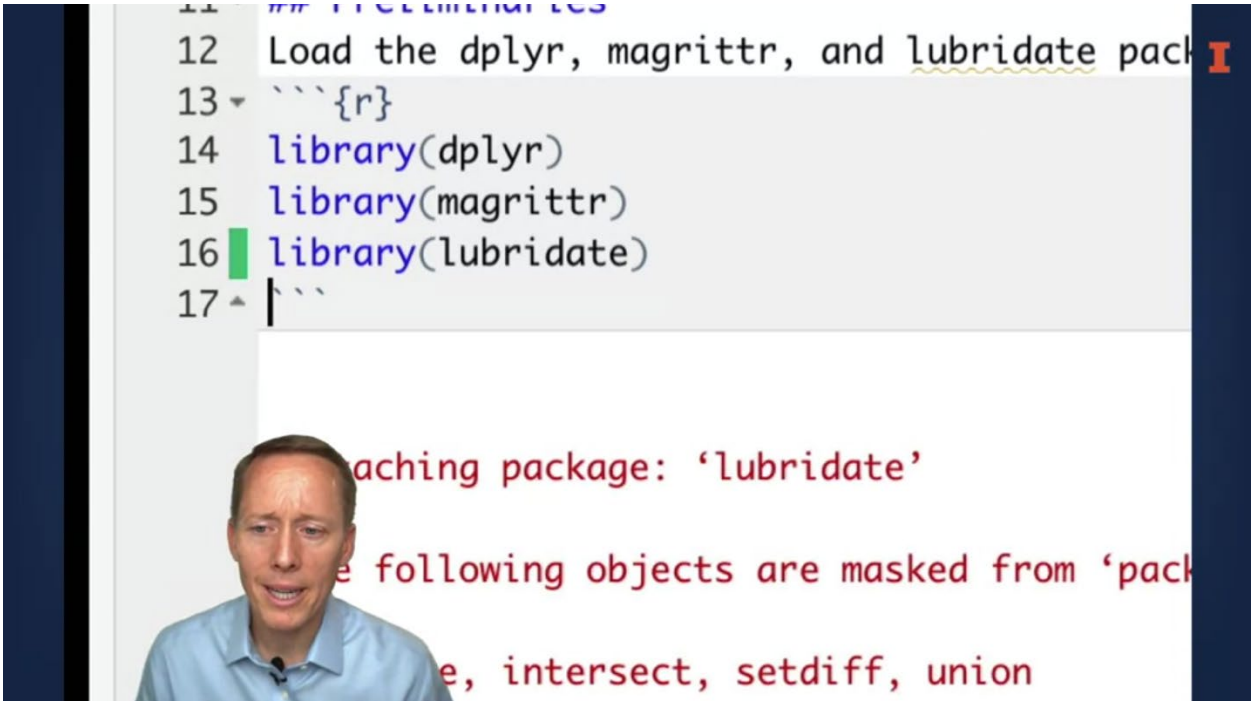
to use the medium. But that's essentially how you can very quickly impute missing values.



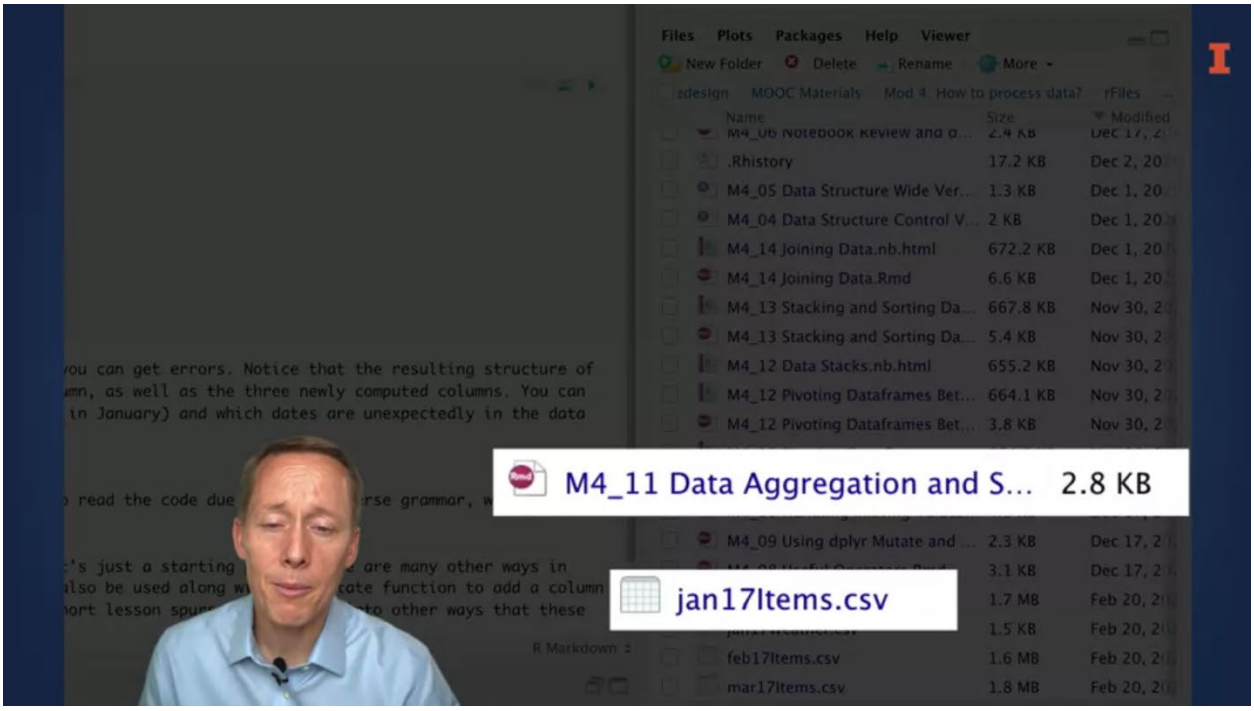
Now this if/else function is very powerful. And another lesson we'll talk about conditional statements in more depth.

Lesson 4-1.11 Data Aggregation and Summary

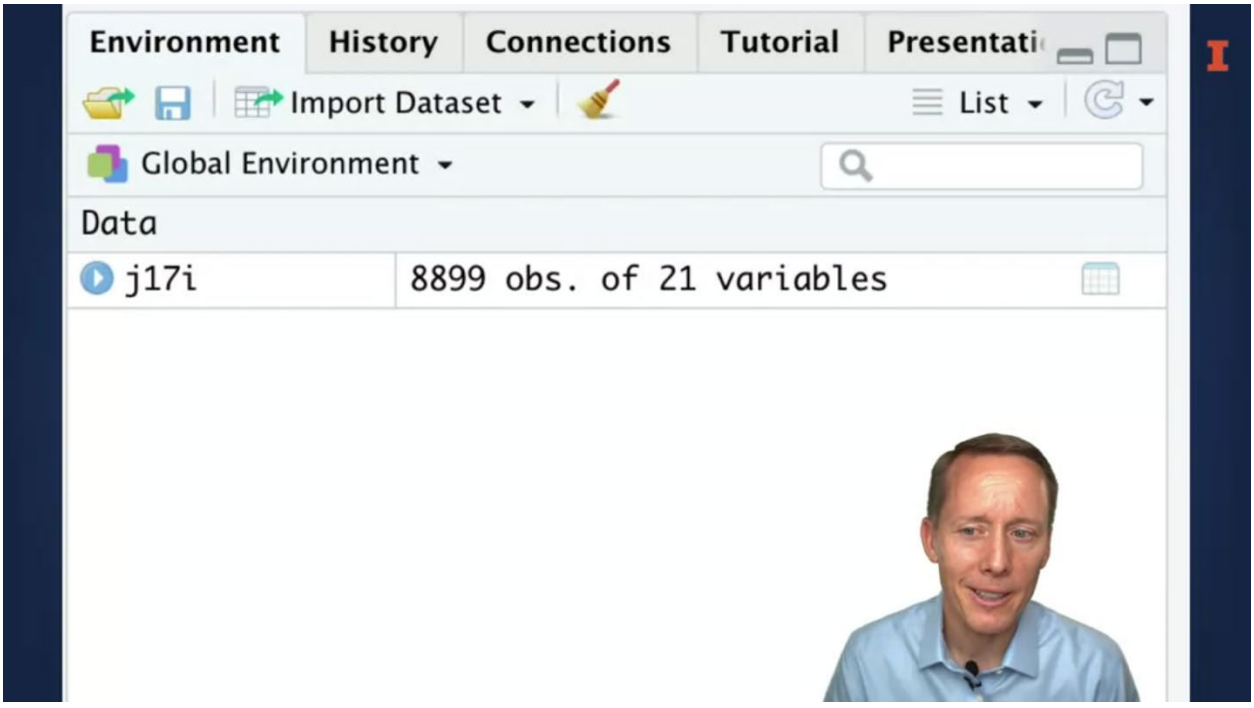
In this lesson, I will introduce two functions for aggregating data; the group by and summarize functions. I will also review how to use the Lubridate package for converting character strings into date-time objects, and then also for rounding date-time values to just date values. Finally, I'll also use the n distinct function for calculating the number of distinct different values.



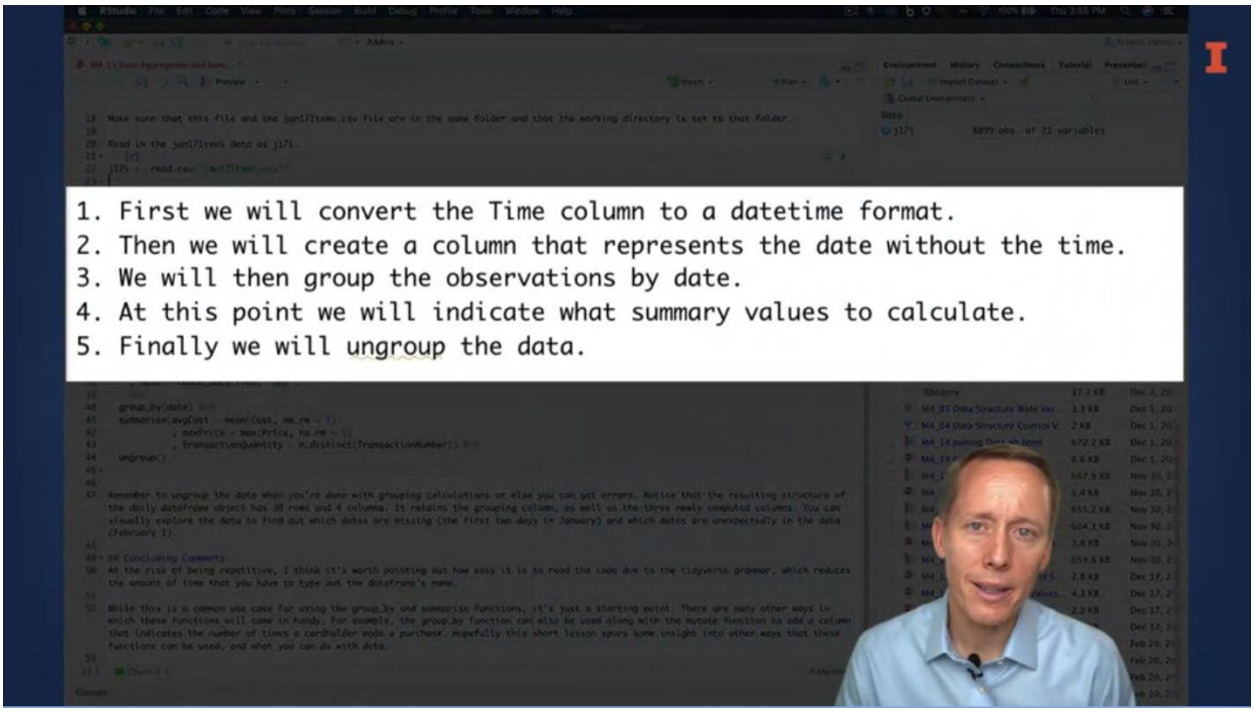
First of all, make sure that you have installed and loaded the dplyr, magrittr, and Lubridate packages.

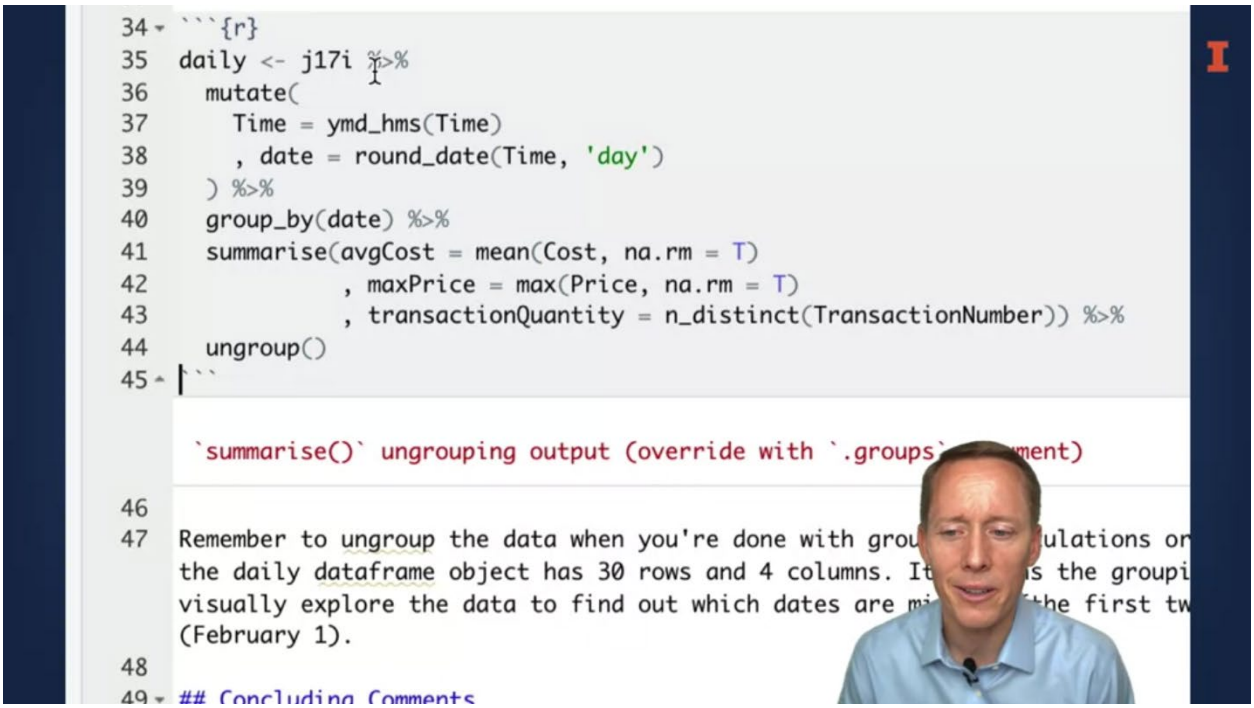


Next, make sure that this file and the Jan17items file are in the same folder and that the working directory is set to that folder.



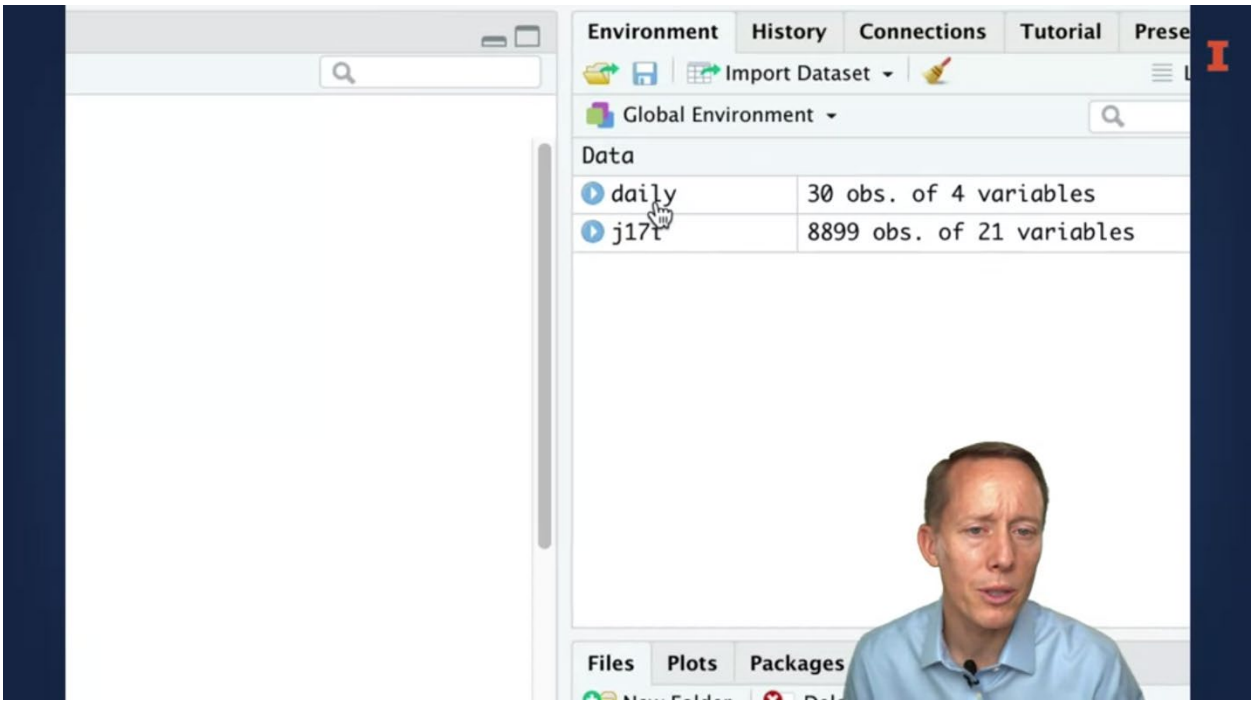
I've already done that, so now I'm going to go ahead and read in the jan17Items.csv file and create a DataFrame object j17i. Now what I want to do is I want to aggregate the data such that I have an observation for each day rather than for every line item of every transaction.



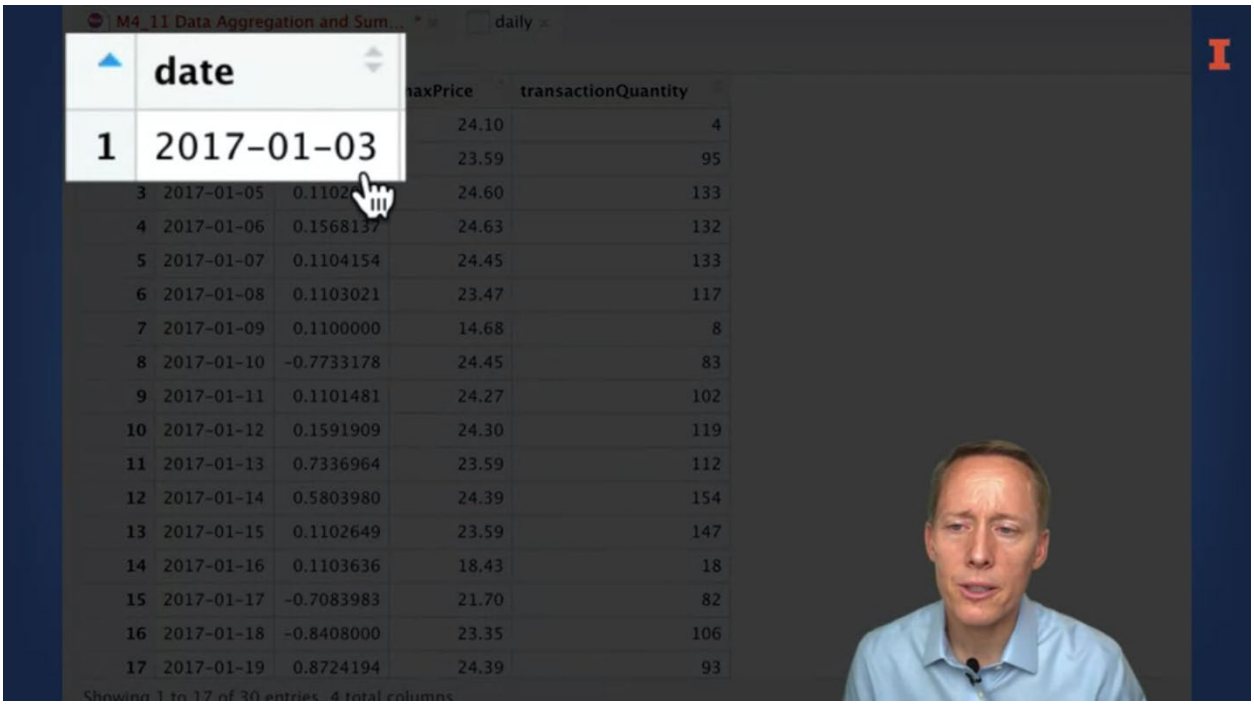


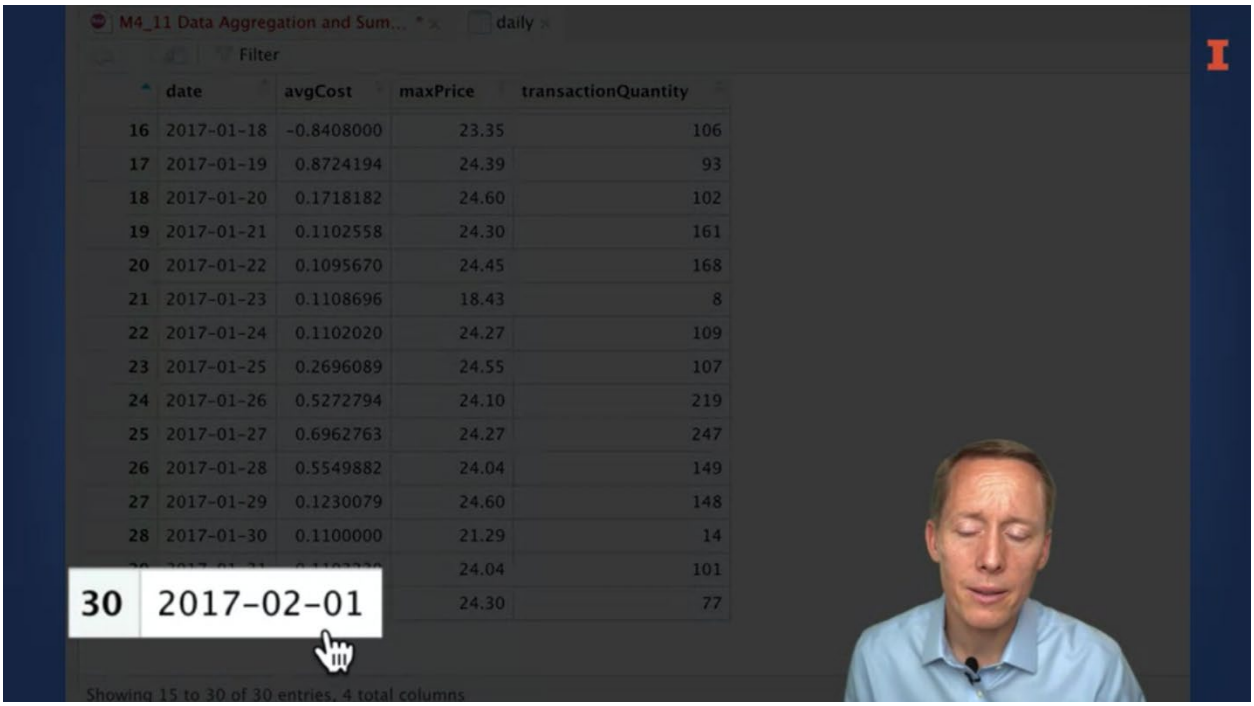
I'm going to do that in five steps here, first I'm going to take the j17i DataFrame and I'm going to convert, I'm going to use the mutate function, so pipe it into the mutate function and create or convert the time column into a date-time object by using the ymd_hms function from the Lubridate package. Now what that will do is it will just convert time from a character string to a date-time type. But I want to aggregate at the day level, the next step is I'm going to create a new column date, and I'm going to use the round_date function from the Lubridate package and I will base it off of the time column and I will

round it to the day. At that point, I will have one new column in my j17i DataFrame, and I'm going to send it into the group by function. I'm going to group by date, I'm just indicating the columns name that I will have created, and at that point, I'll pipe it into the summarize function, where I will aggregate the data and the additional columns that I want are the average cost and I'll calculate that by calculating the mean of the cost and I'll make sure to not include missing values, then I'll also calculate the maximum price for each day by using the max function on the price column, and again, make sure to explicitly indicate that I don't want to include missing values, and then I want to identify how many unique transactions occurred each day, so I'll create a new column transaction quantity, and that will be based off of this n distinct function. Basically, we're going to say, "Hey, put the transaction number column in there and calculate how many different distinct transaction numbers there are." Then the last step is I'm going to ungroup the data, I'll go ahead and run that, and by the way, ungrouping the data is really important, if you don't do that, it can cause some errors later on.



Let's go ahead and look at this daily DataFrame object, you can see there are 30 observations and four variables.



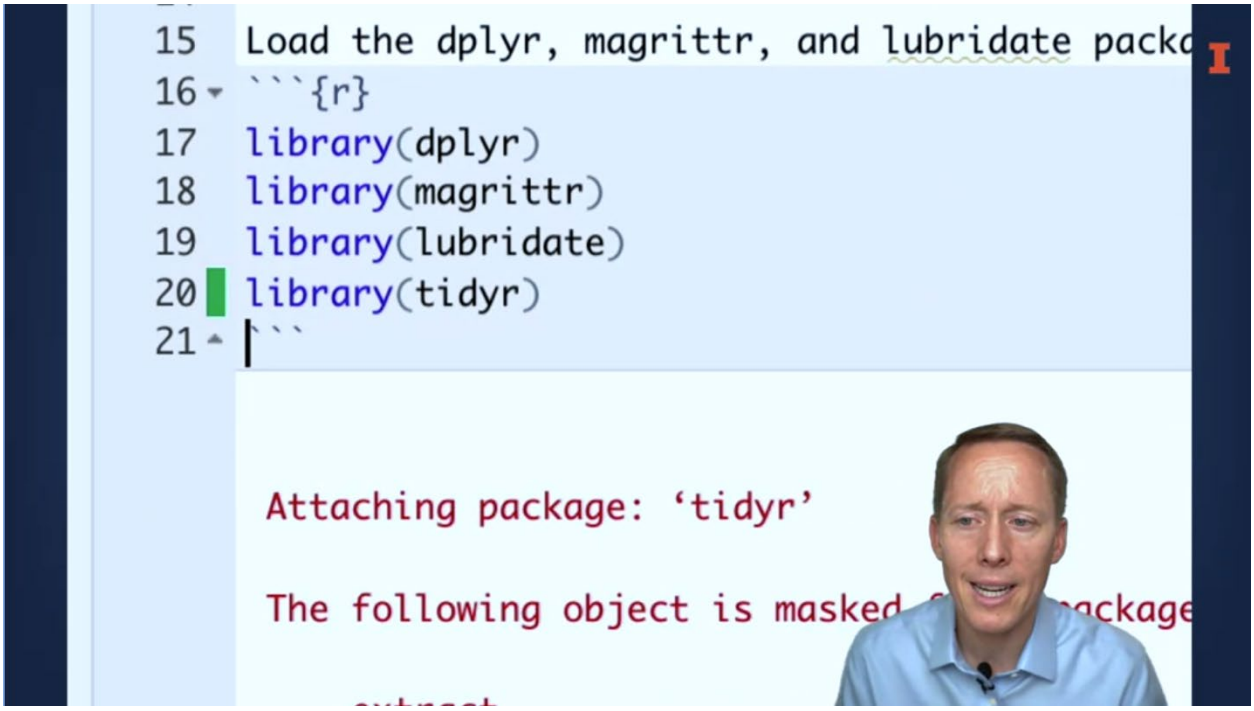


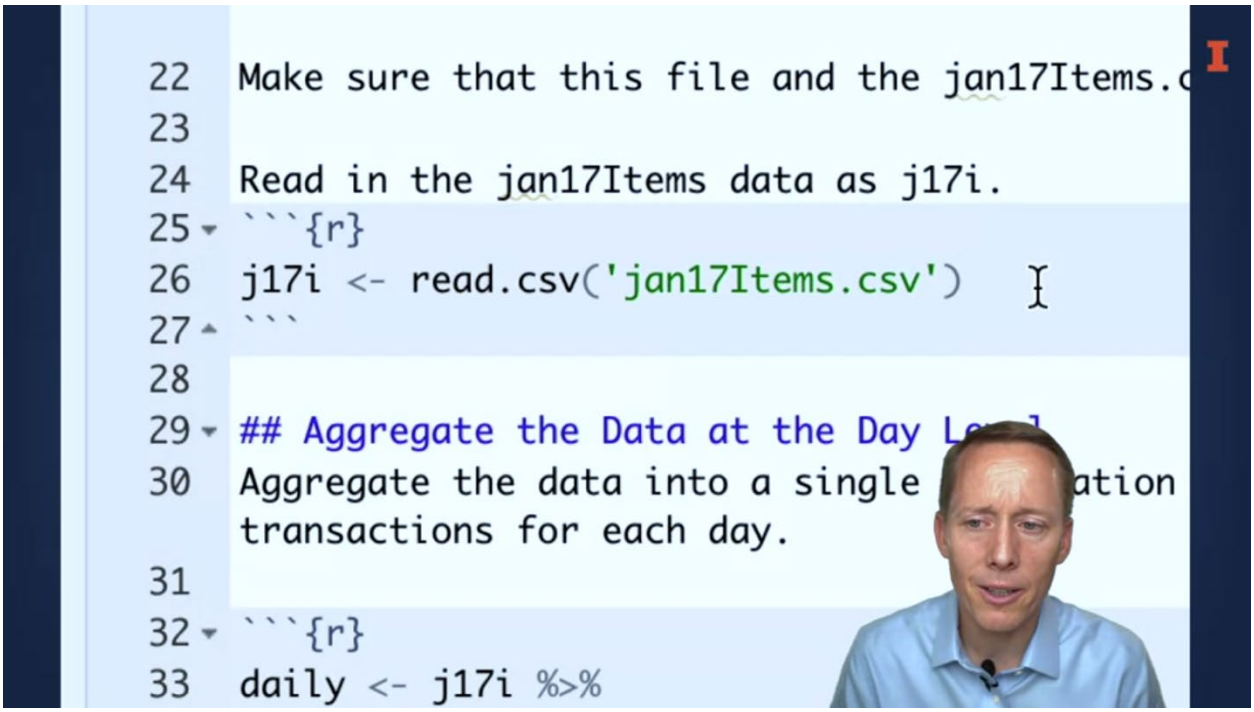
If I visually explore that, you can see that it starts at January 3rd, and I could go down and make sure that every date is included in here. You might wonder, why is it only 30 observations when there are 31 days in January? Well, it is pretty apparent here that I'm missing the first and second, but I'm also including some observations from February 1st. Anyway, that's why we've got 30 observations in there, but you can see that it tells me the average cost for each day, the maximum price, and the transaction quantity.

I hope that example is a good starting point for aggregating data using the group by and summarize functions, it's very easy to do, and at the risk of being repetitive, I just want to point out how easy it is to read what's going on here without using comments. Again, that is a strength of using The Tidyverse Grammar, and that is just a starting point for using the group by and summarize functions. You can also use group by and mutate together, for instance, if you want to calculate the number of transactions that each cardholder makes.

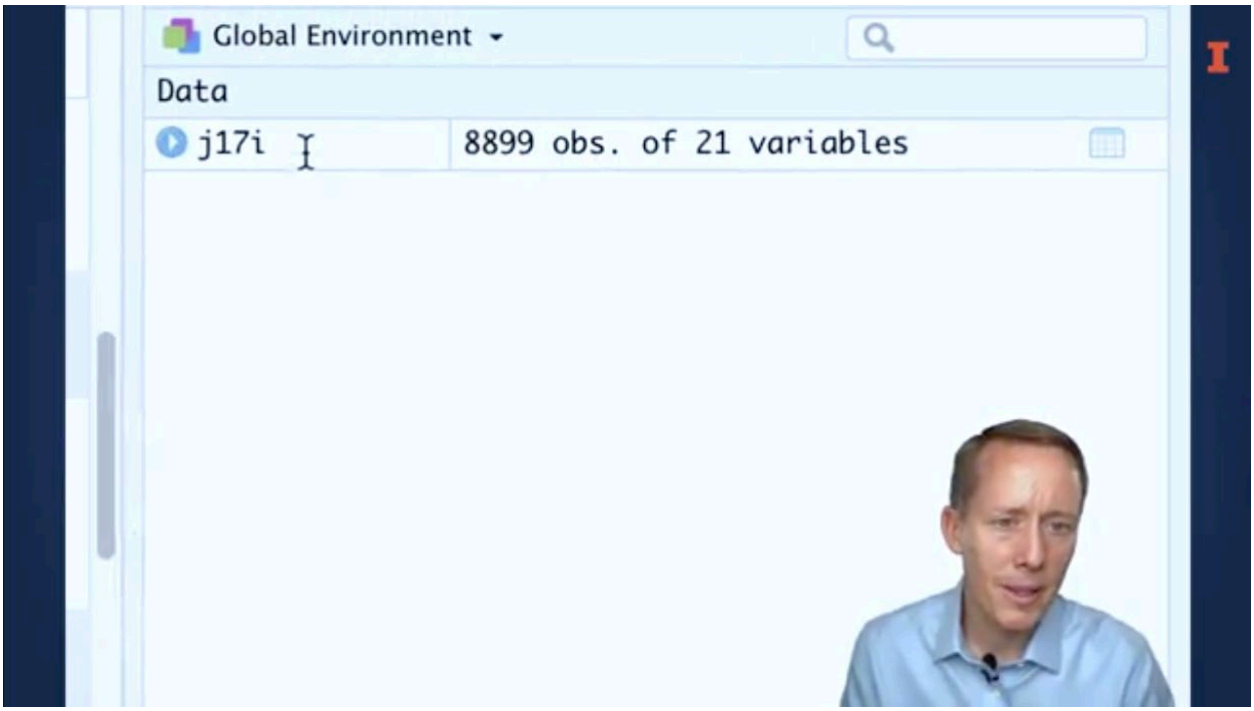
Lesson 4-1.12 Pivoting Dataframes Between Wide and Long Shapes

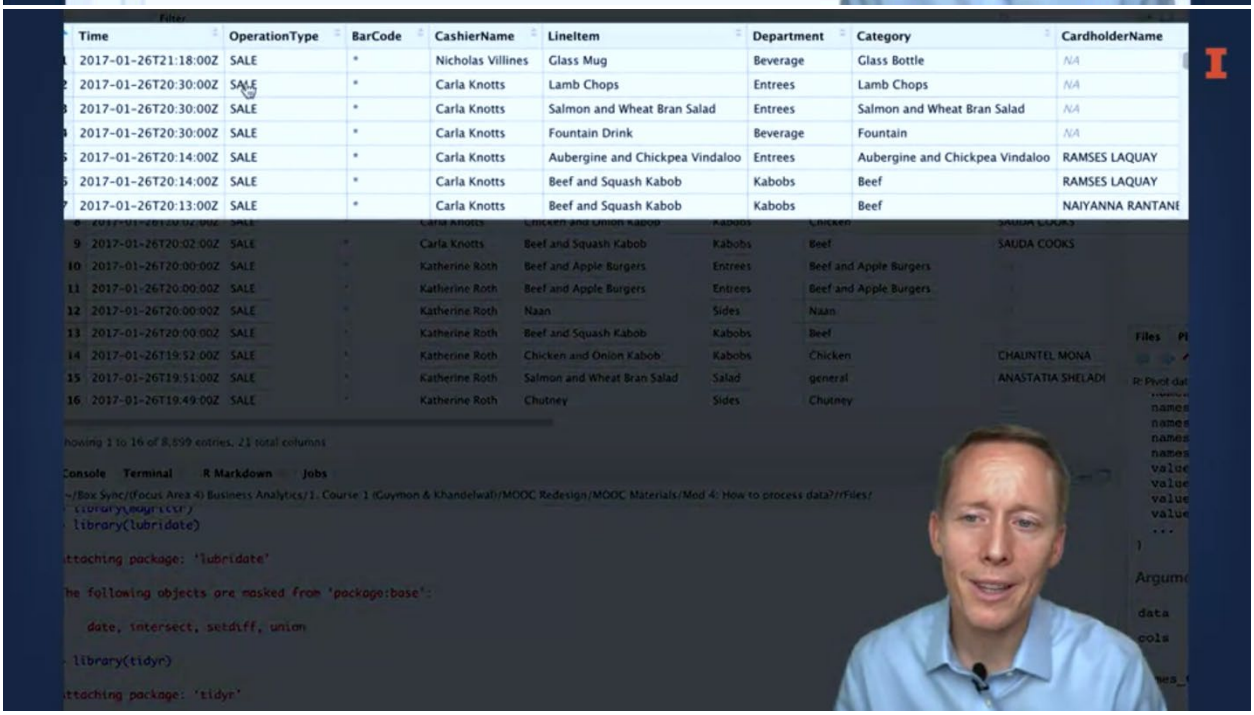
In this lesson, I want to introduce two functions for pivoting data between wide and long shapes. These functions pivot wider and pivot longer are from the Tidier package, which is one of the packages in the tidyverse and contains a lot of functions for reshaping data. Now briefly, the shape of a data frame refers to the number of rows and columns in the data frame. A long data frame has a lot of rows a wide data frame has a lot of columns.



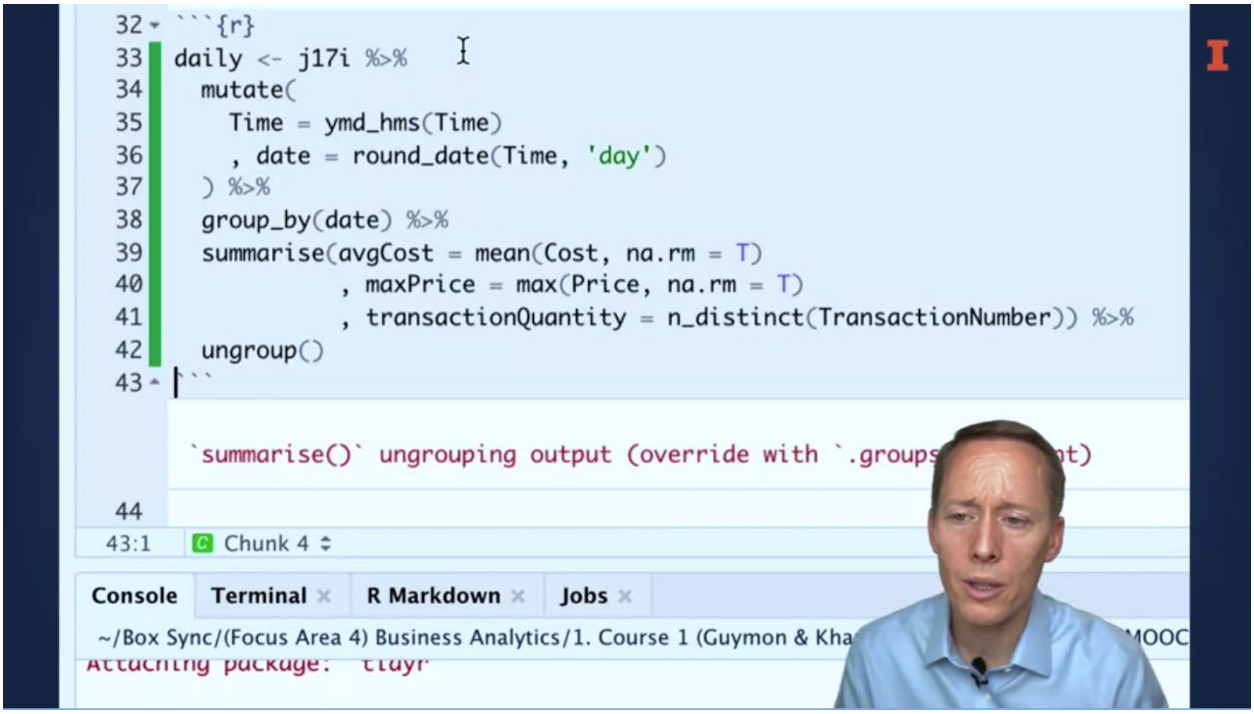


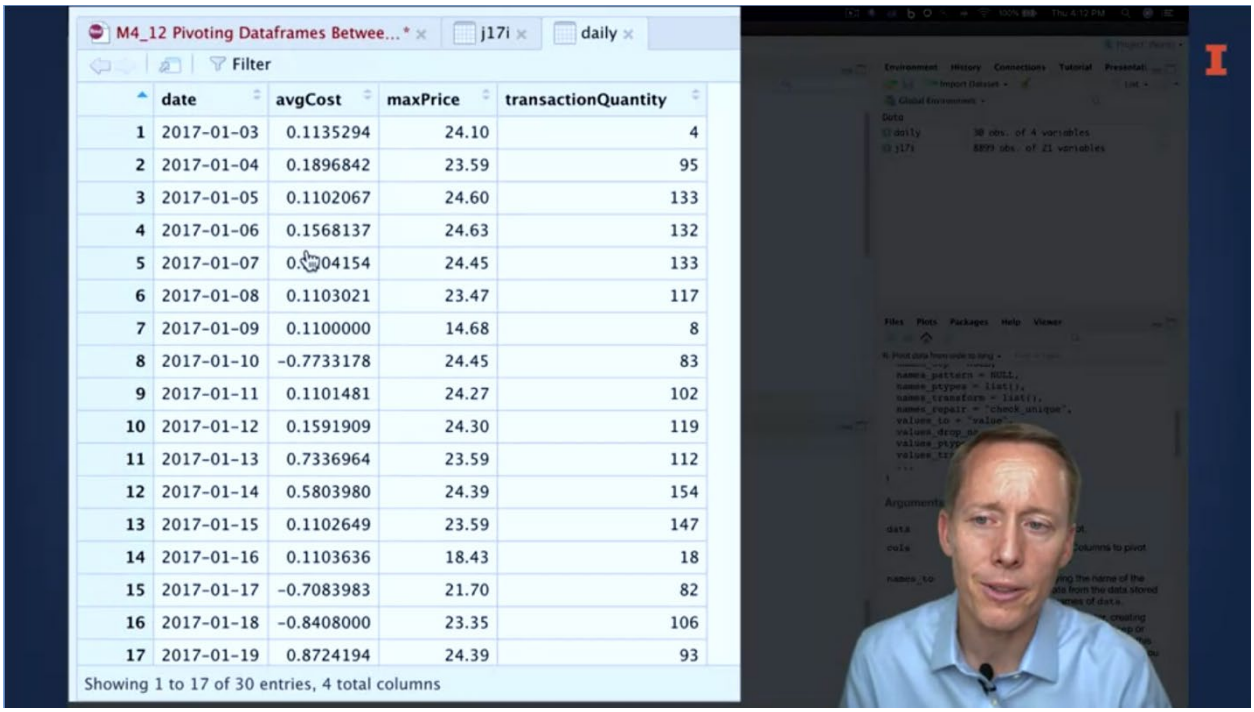
So to illustrate why we might want to do this, let's first make sure that we have installed the tidyr package. So if you haven't already done so, then run the install that package is for tidyr. And then make sure and load the dplyr, magrittr, lubricate, and tidyr packages so that you can access those functions. And then also read in the Jan17 items.Csv file as the J79 data frame object.





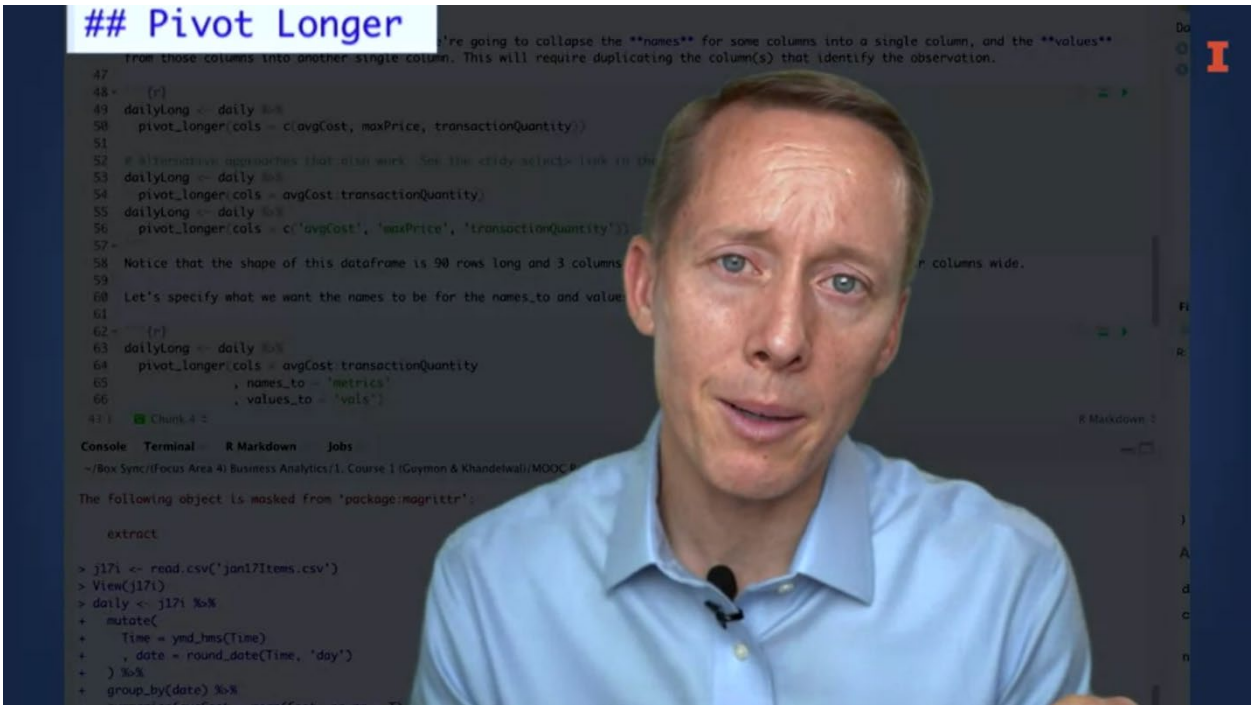
At this point, we've got a data frame that has 8899 observations, and each observation is one line item of one transaction. So this is a lot of detail about what went on in the month of January to illustrate longer and wider data frames.



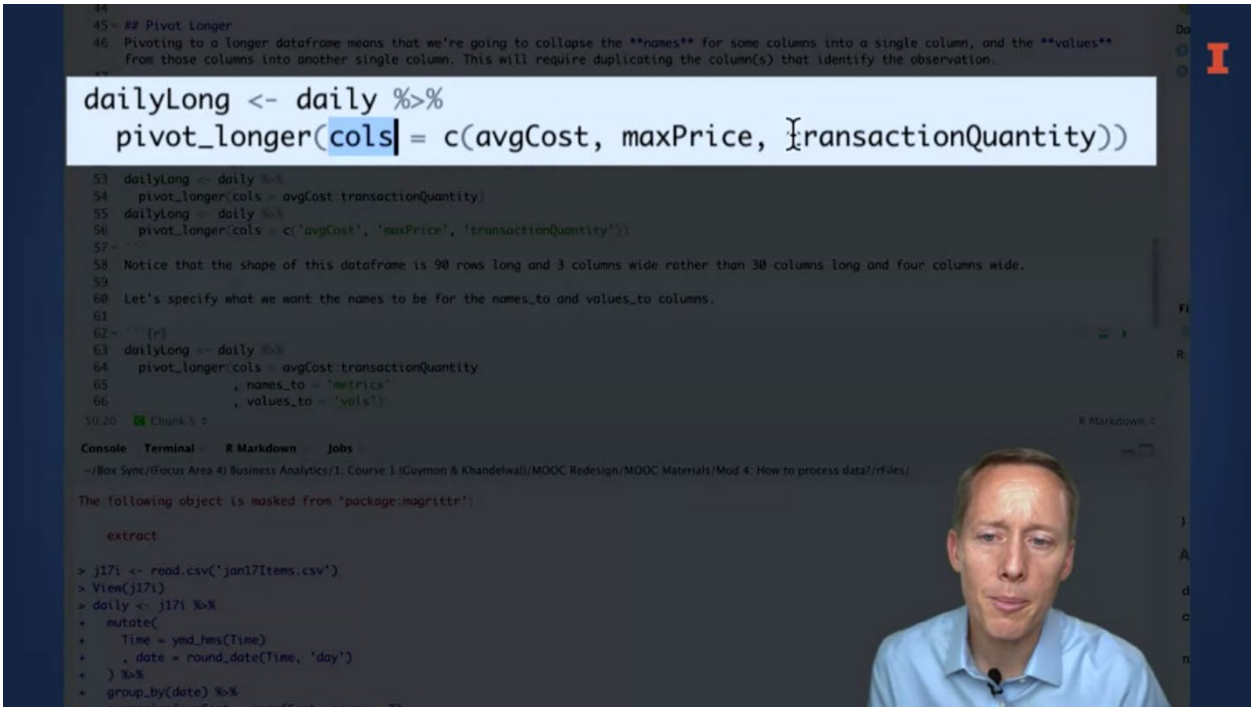


	date	avgCost	maxPrice	transactionQuantity
1	2017-01-03	0.1135294	24.10	4
2	2017-01-04	0.1896842	23.59	95
3	2017-01-05	0.1102067	24.60	133
4	2017-01-06	0.1568137	24.63	132
5	2017-01-07	0.1104154	24.45	133
6	2017-01-08	0.1103021	23.47	117
7	2017-01-09	0.1100000	14.68	8
8	2017-01-10	-0.7733178	24.45	83
9	2017-01-11	0.1101481	24.27	102
10	2017-01-12	0.1591909	24.30	119
11	2017-01-13	0.7336964	23.59	112
12	2017-01-14	0.5803980	24.39	154
13	2017-01-15	0.1102649	23.59	147
14	2017-01-16	0.1103636	18.43	18
15	2017-01-17	-0.7083983	21.70	82
16	2017-01-18	-0.8408000	23.35	106
17	2017-01-19	0.8724194	24.39	93

I want to aggregate the data at the daily level. So I'm going to run this code chunk here to create this daily data frame that now just has one observation for each day in this data set in the J 17 I data frame so you can see it. It starts at January 3rd 2017 and ends on February 1st 2017, and we have average cost max price and transaction quantity.

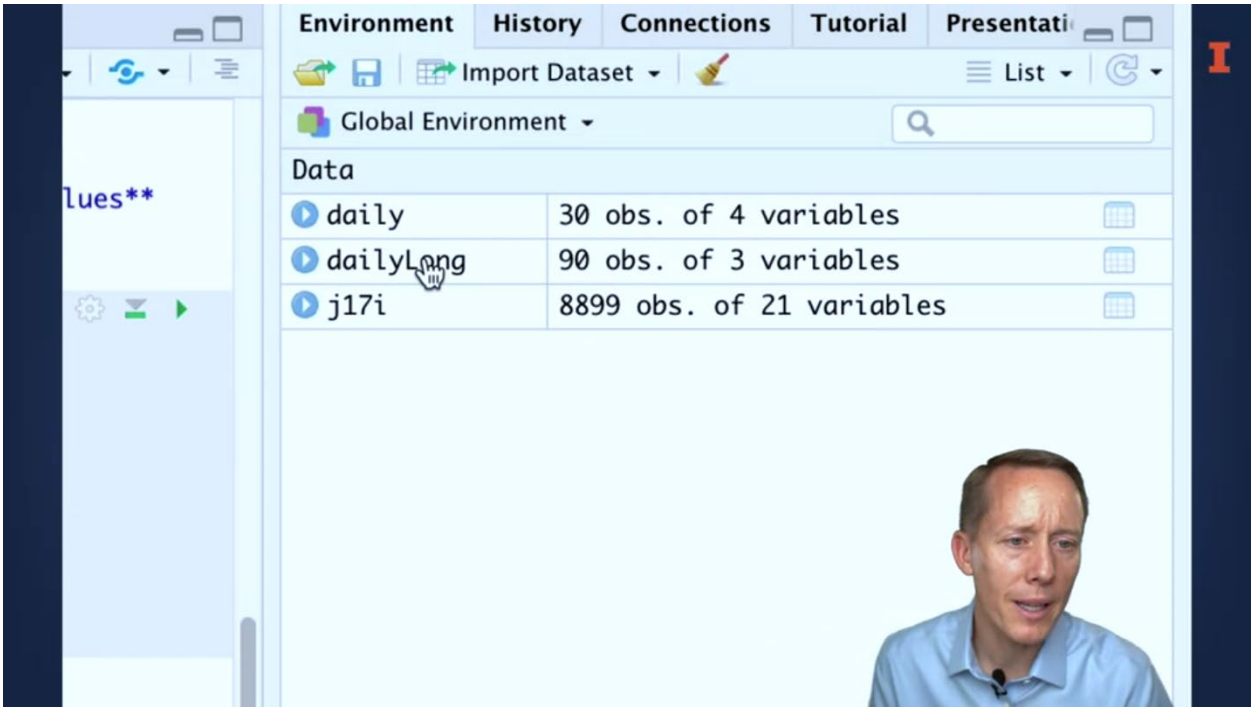


So at this point, we can talk about pivoting the data. To pivot the data longer means we're going to take some of the column names and collapse those into a single column. Then the values from those columns will put into a different single column. That means we'll have to duplicate the date many times.

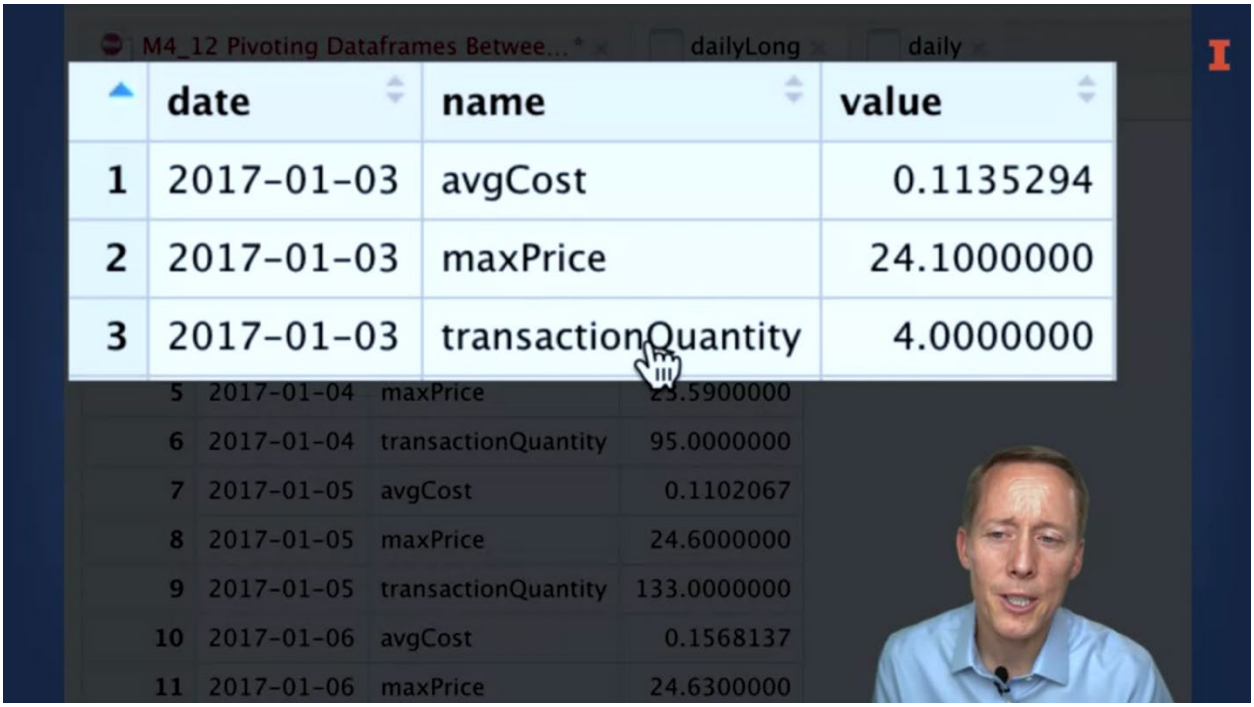


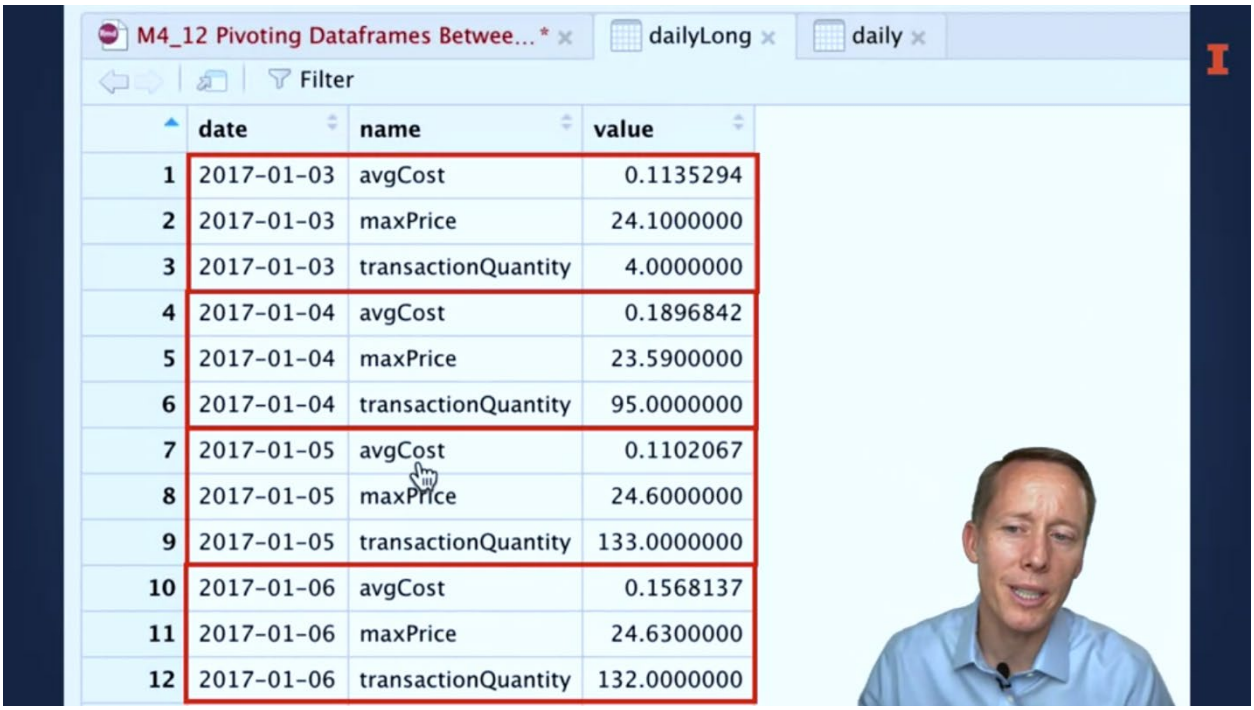
So here's how we'll do that. We'll take the daily data frame and we will pipe it into the pivot longer function. Now, this first argument here calls indicates which columns we

want to convert into just two columns, one that has the column names one has the values. And we're going to take the average cost, max price, and transaction quantity, and we'll create this new data frame called Daily Long.

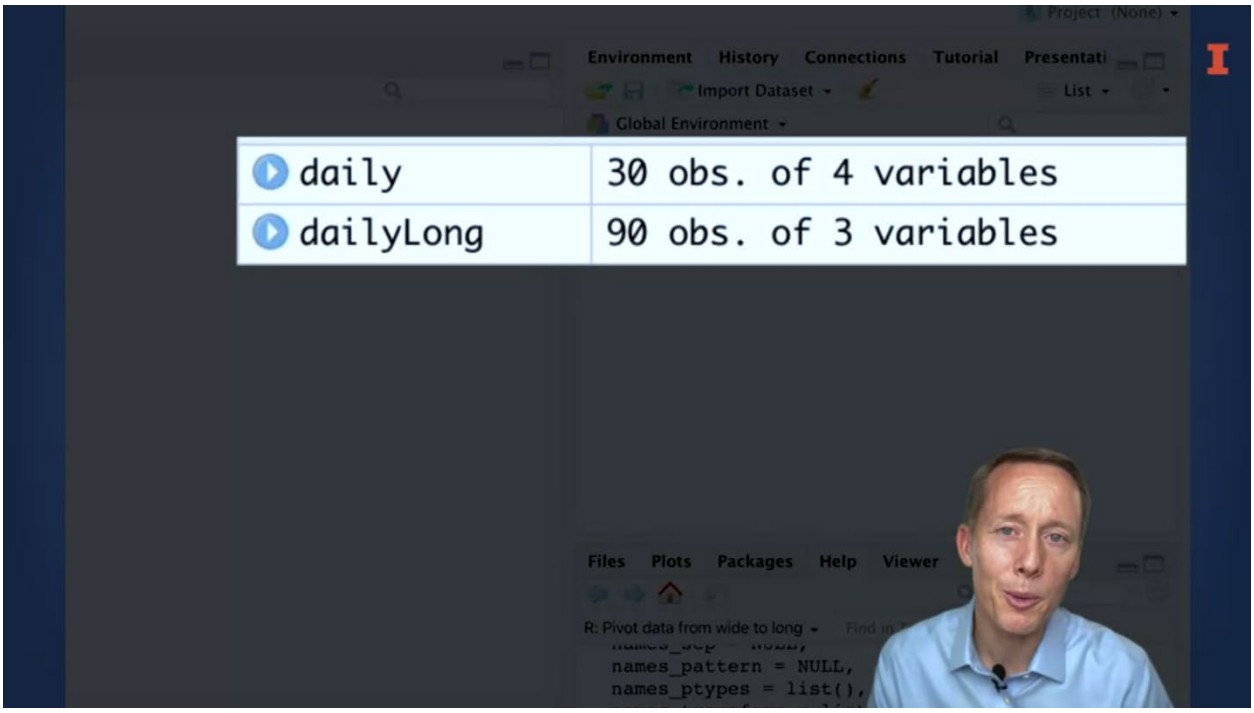


Let's go ahead and look at that data frame, and we can see that it has 90 observations of three variables.

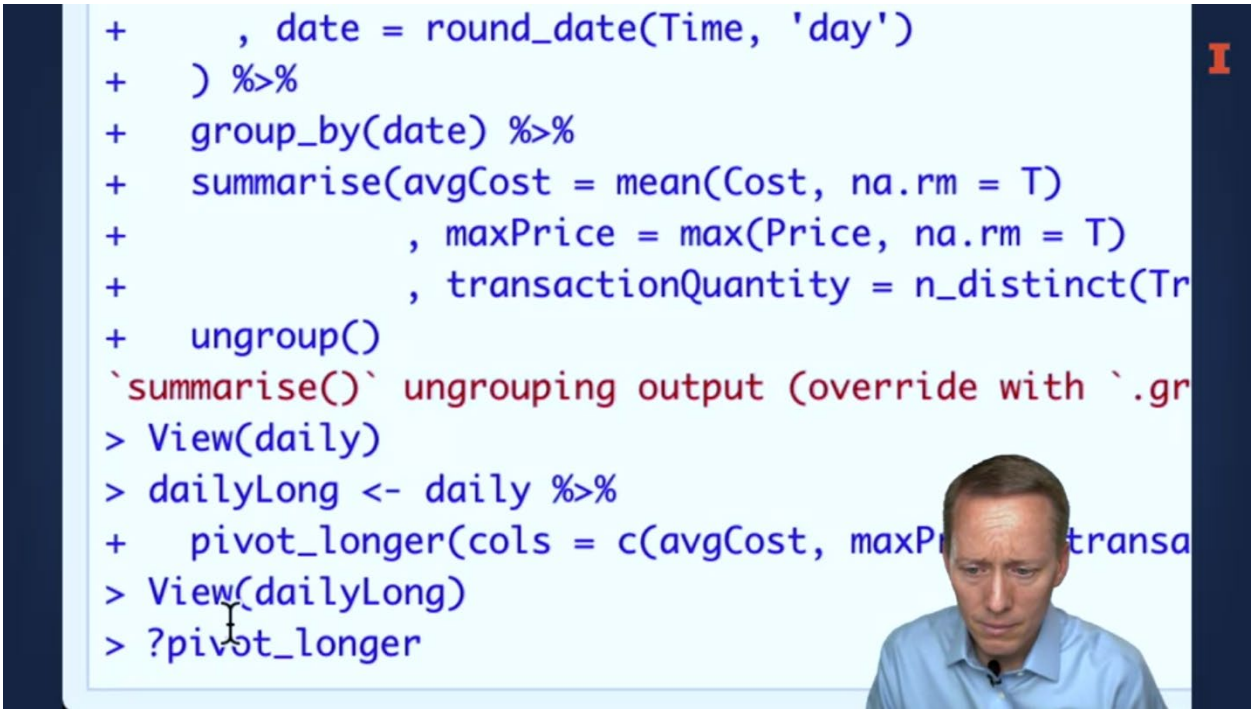




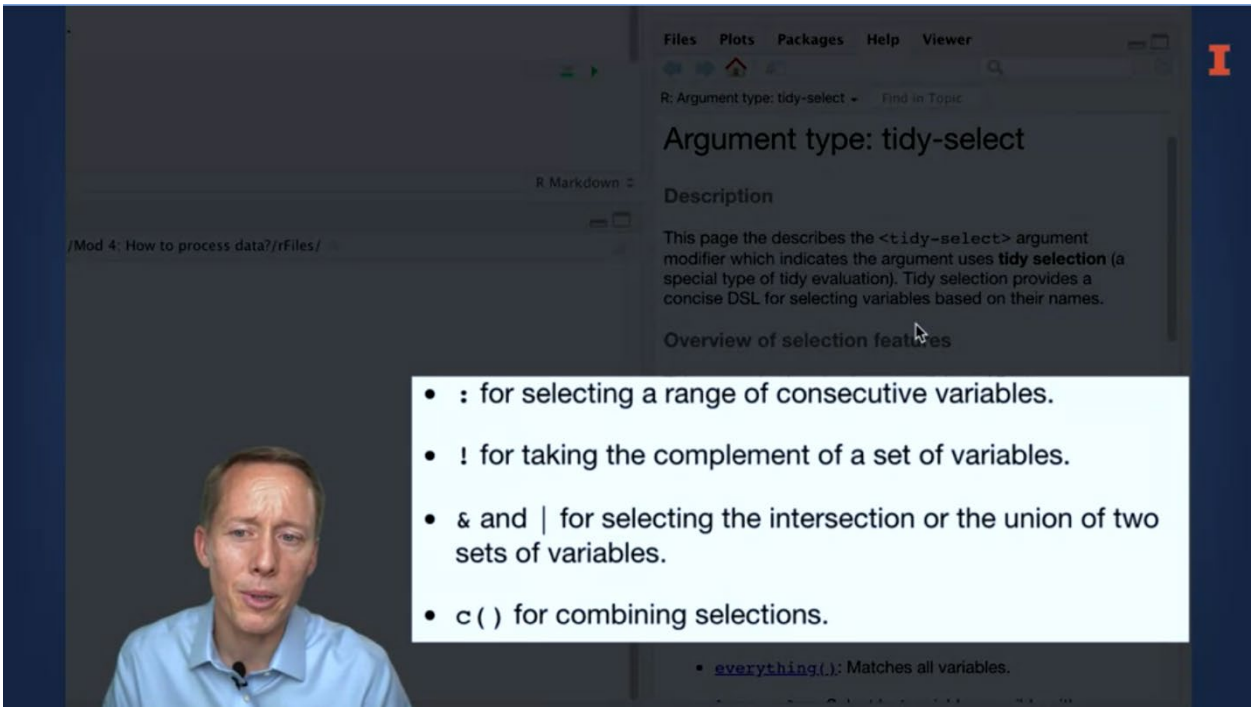
And as we visually explore it, we can see that for January 3rd, we've got three different observations. And in the name column, we've got the average cost, max price and transaction quantity, and you can see that repeats for each day. In the value column, we have the values that pertain to each of those values in the name column.



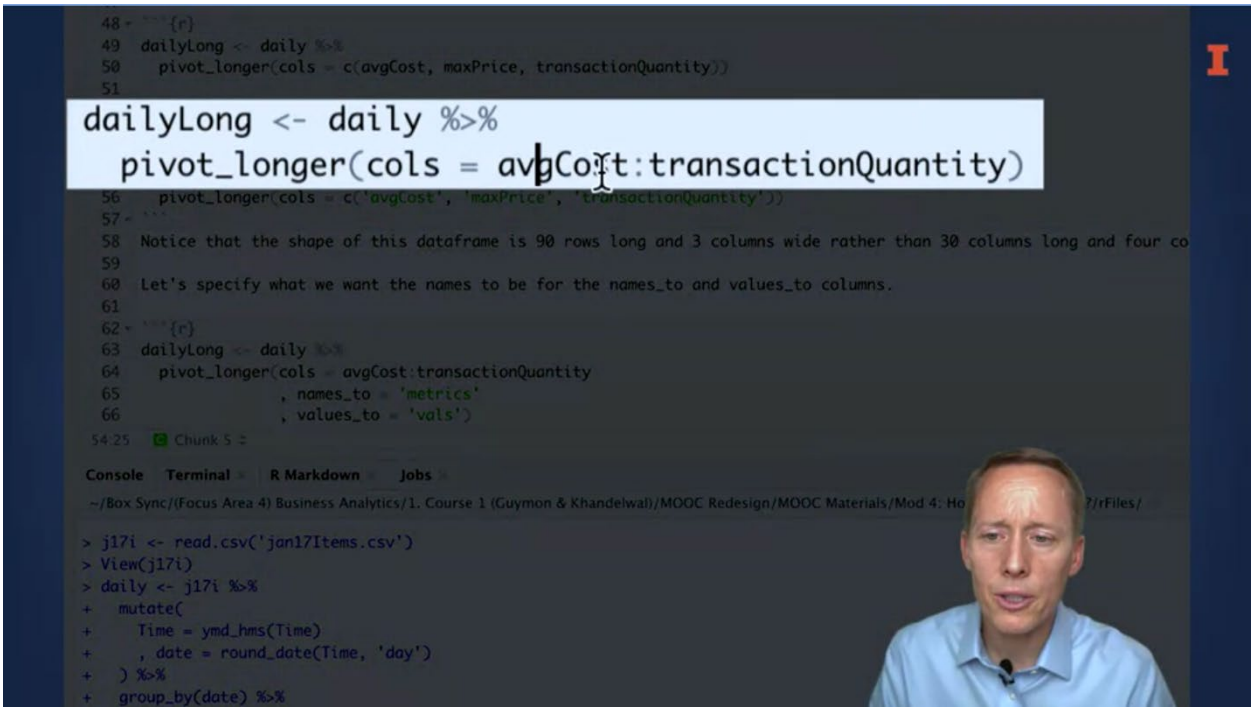
All right, so you can see how this is a longer data frame and why it's 90 rows instead of 30 rows. We've replicated each date three times.

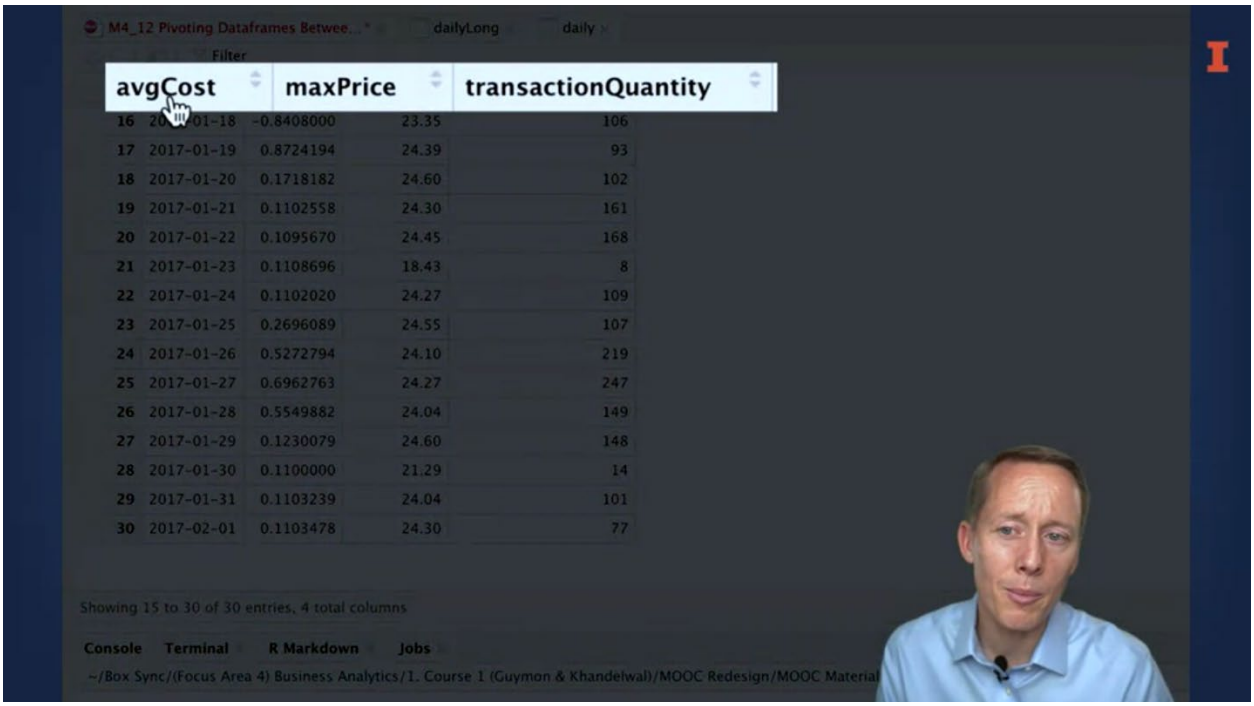


Now, if you read the documentation for pivot longer.

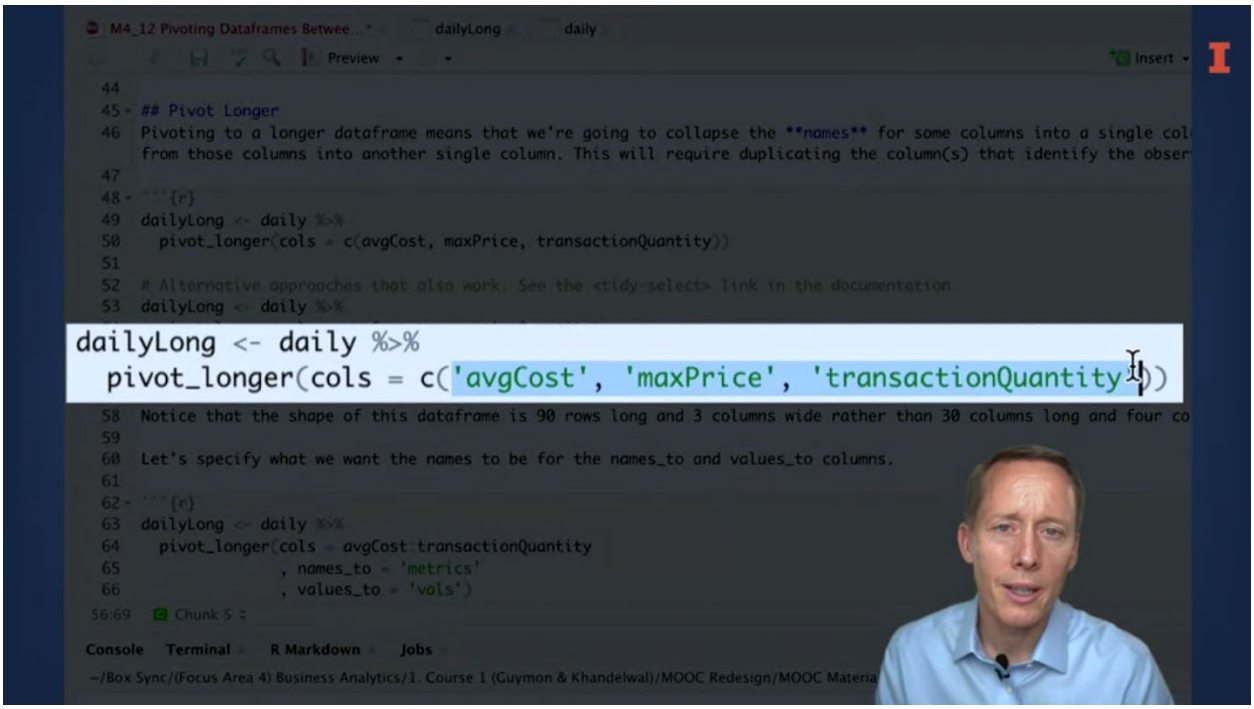


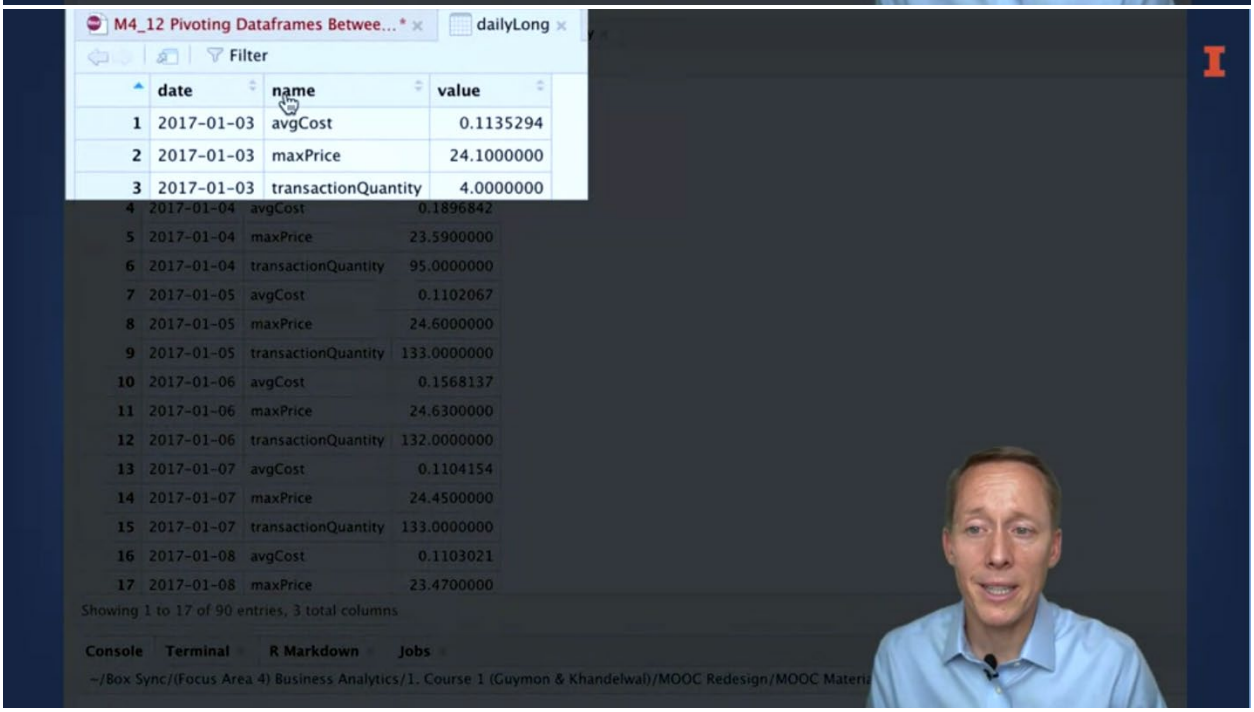
You can see that in this calls argument here there's this tidy select link, and if you click on that, it tells you different ways that you can select columns that you want to pivot.



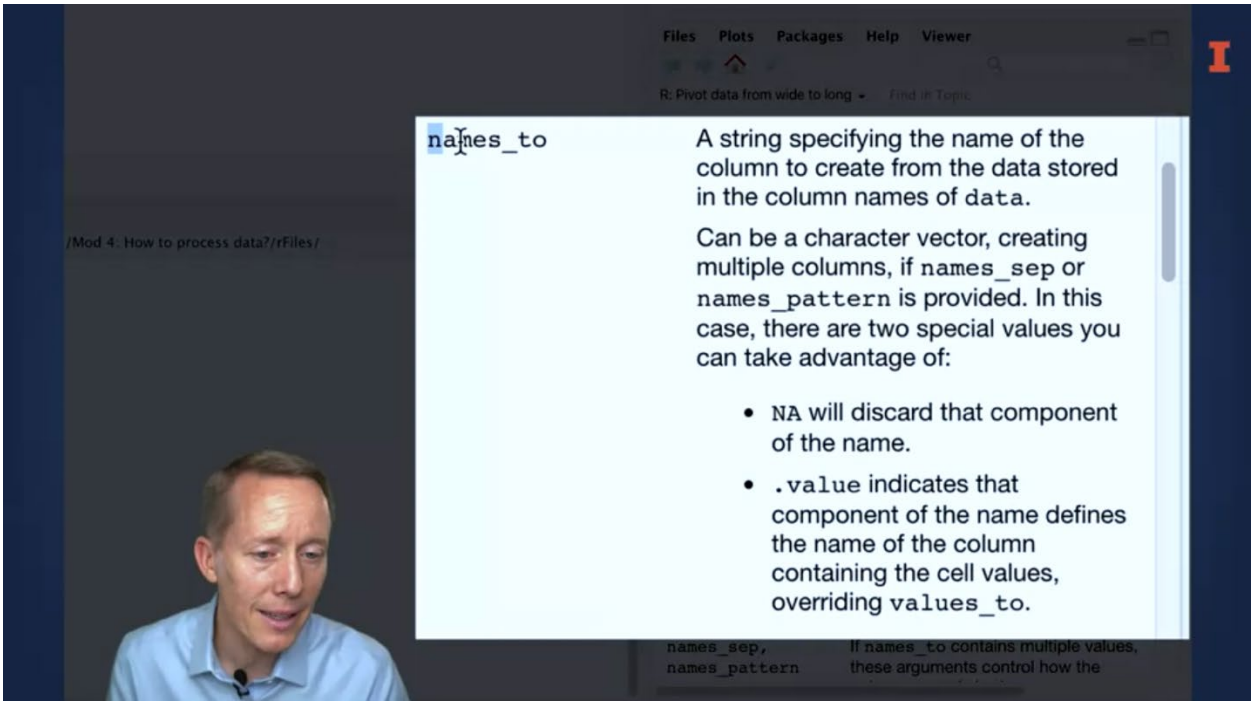


So you can also perform the same thing here by using average cost: transaction quantity, to indicate that we want to take everything from average cost to transaction quantity.

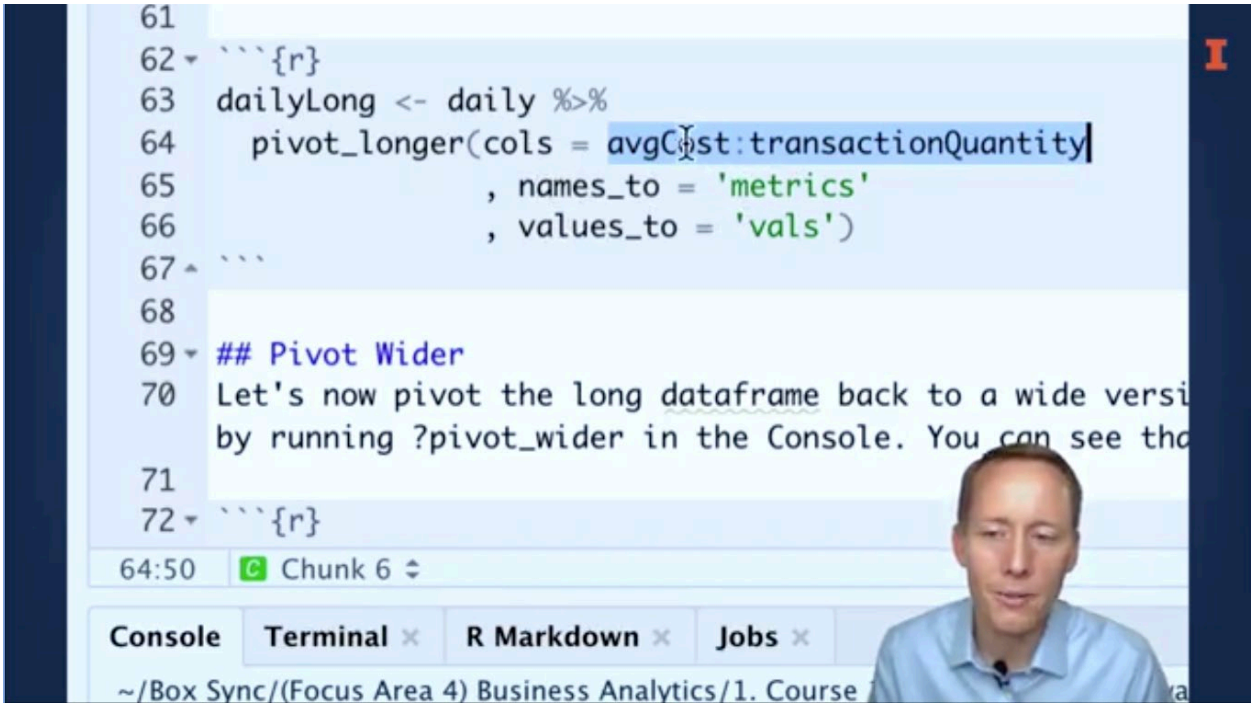




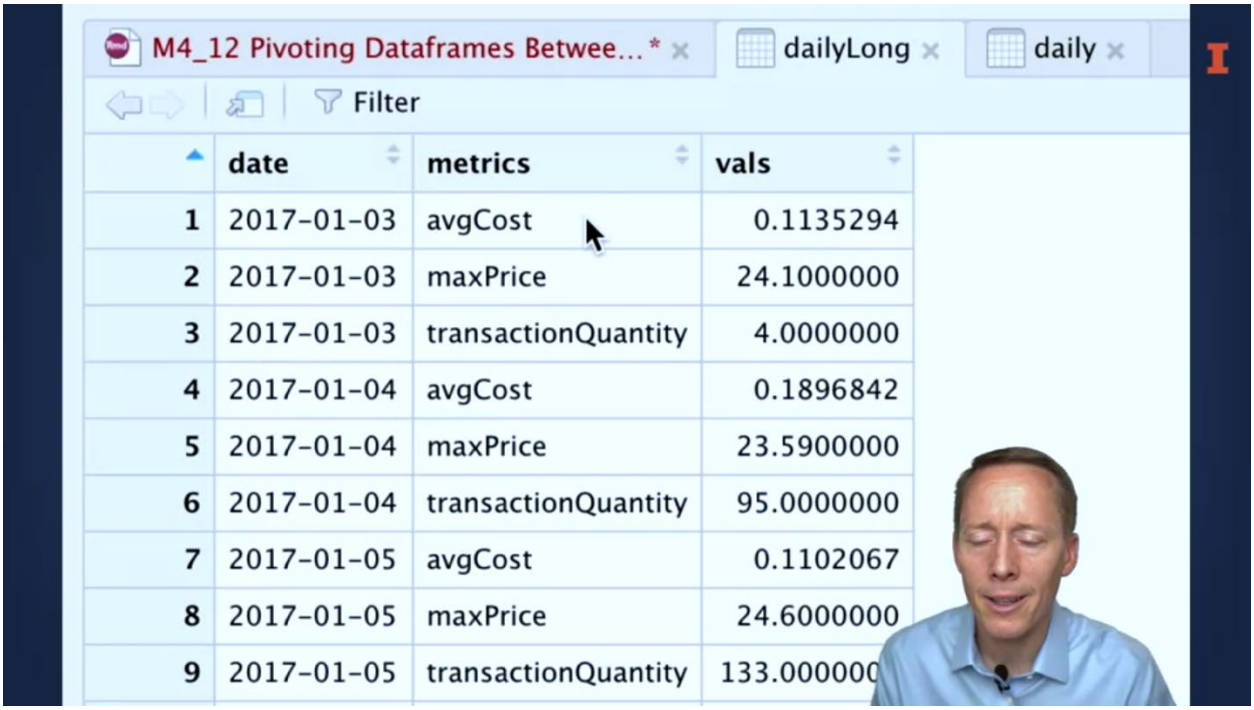
And you can also use kind of the base our approach, where you put quotation marks around the column names and put that into a vector, okay? Either way, you'll get the same thing, but it might be worth looking at if you're dealing with very wide data sets. Now, let's go back to this daily long data frame and recognize that it, by default, created new column names for name and value.



If we read the documentation for the pivot longer, we can see that there are additional arguments that we can include. So that we can change these column names to something that is more meaningful to us.

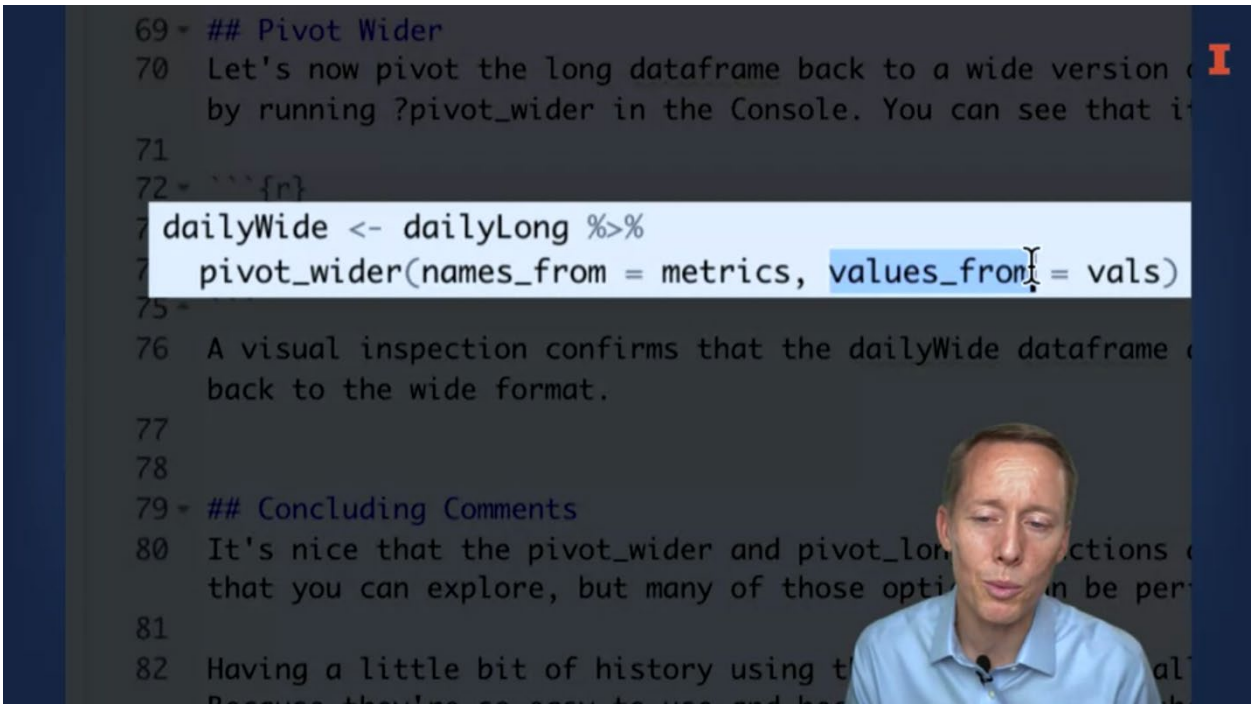


So let's go ahead and run this, and we'll use this average cost: transaction quantity just to prove that it gives us the same thing. And will change the names column to metrics and the values column to vals.

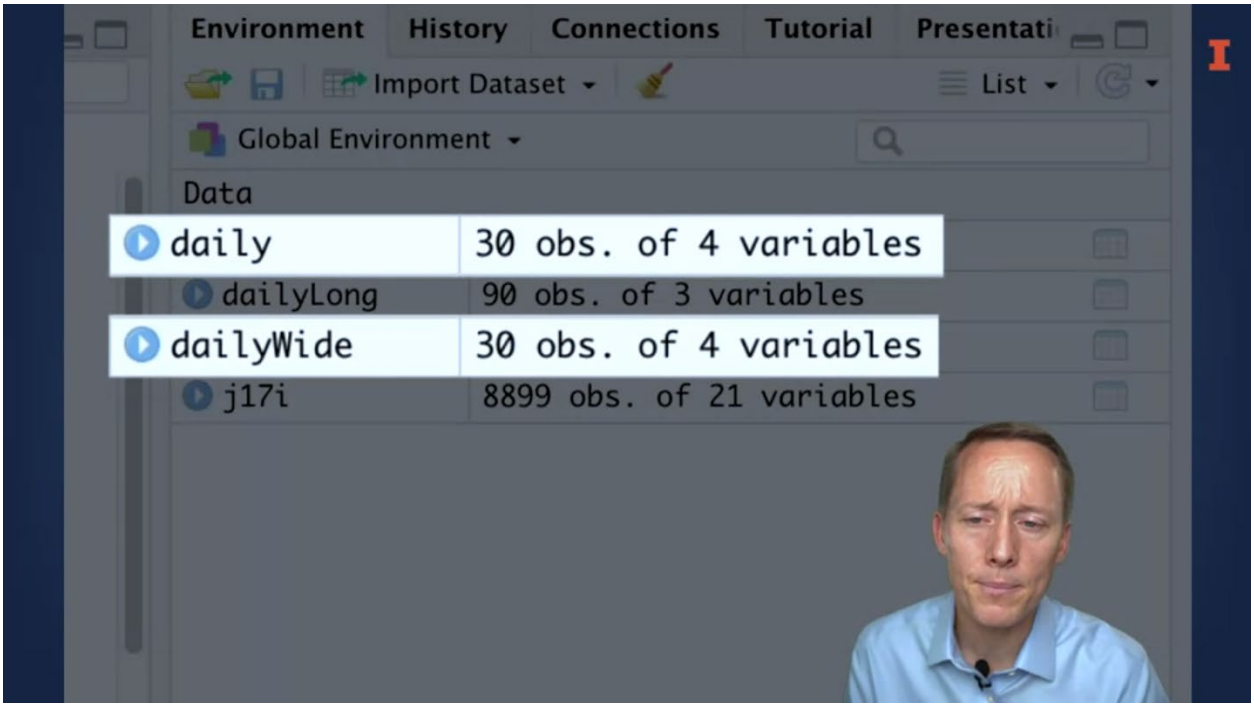


	date	metrics	vals
1	2017-01-03	avgCost	0.1135294
2	2017-01-03	maxPrice	24.1000000
3	2017-01-03	transactionQuantity	4.0000000
4	2017-01-04	avgCost	0.1896842
5	2017-01-04	maxPrice	23.5900000
6	2017-01-04	transactionQuantity	95.0000000
7	2017-01-05	avgCost	0.1102067
8	2017-01-05	maxPrice	24.6000000
9	2017-01-05	transactionQuantity	133.000000

So if I run that look at daily long now, we can see that the second column is metrics the third column is vowels. And if you were to compare the values in the state of frame, they'd be exactly the same. So there's how you can pivot from wide to long.



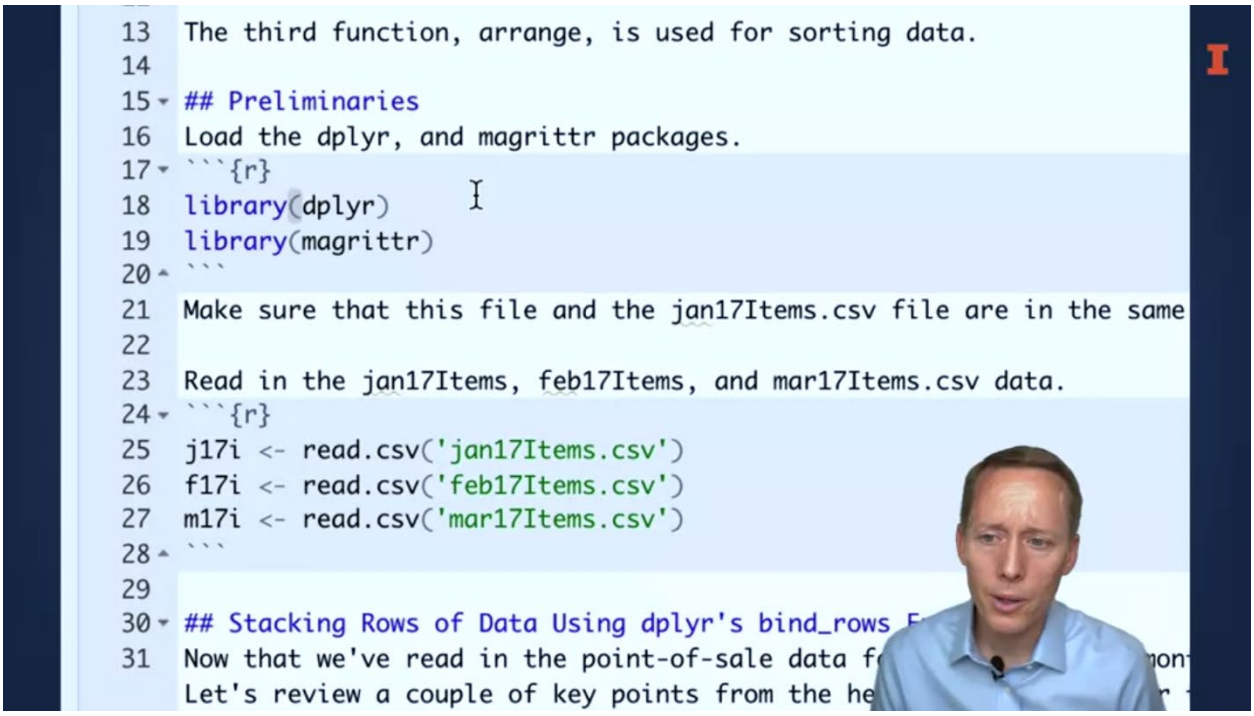
Now let's talk about how you can pivot from long to wide. So fortunately, the tidyverse grammar is very parallel here, so we'll start with the long data frame and we'll feed it into a function called pivot wider, makes sense? And in this case, we need to identify the column from which the names that will be used for the new column names will be created. And that will be the metrics column. And then we need to also identify the name of the column from which the values will be created. And in this case, the values will be coming from the vals column, all right?



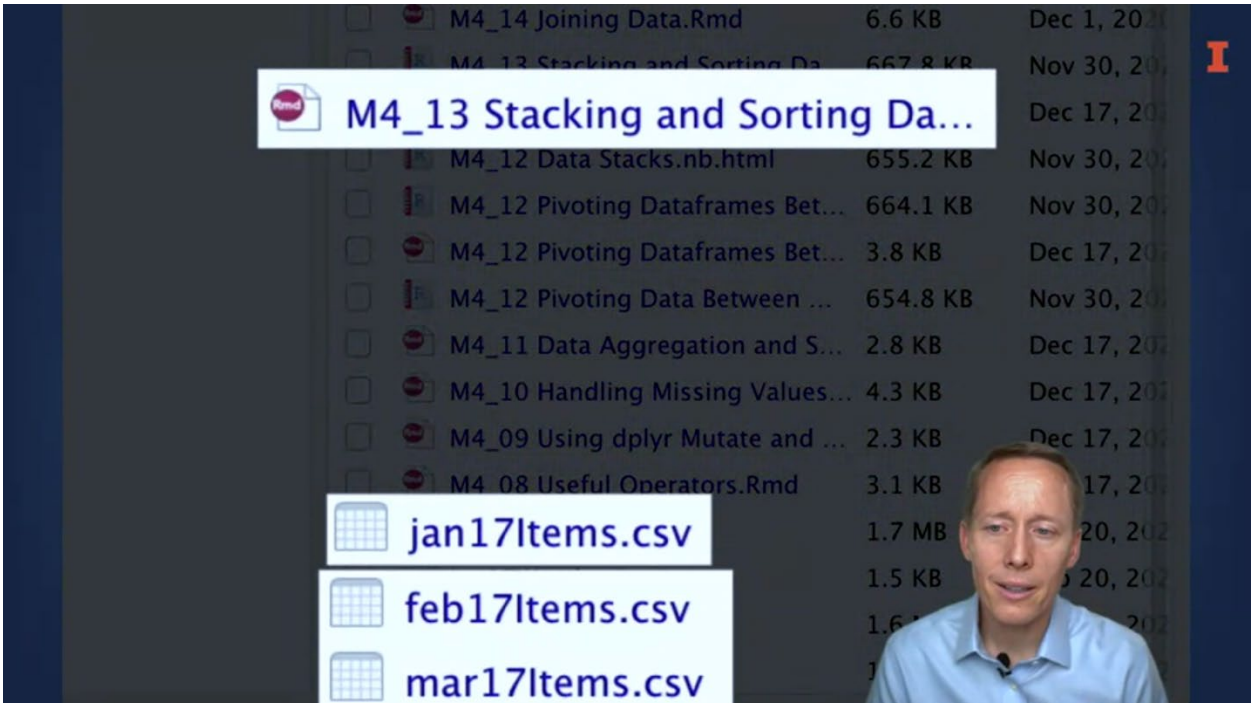
So we'll go ahead and run that, and we've got this new daily wide data frame object. You can see it's got the same dimensions as the daily data frame. And if we compare it to the daily data frame, you can see that they're exactly the same. Okay, so we just basically reverted back to that wide format that we started with. So it's nice that these two functions pivot wider and pivot longer, are very parallel in their usage. I've got a little bit of history using the tidyverse, and they did not used to be parallel like that. They were a gatherer and spread. And anyway, I find it very easy to use the names are so parallel. And because pivoting data has done so frequently and these functions are so easy to use, I consider these functions to be essential functions for anyone who's going to wrangle, and reshape, or pre-processed data with R.

Lesson 4-1.13 Stacking and Sorting Data

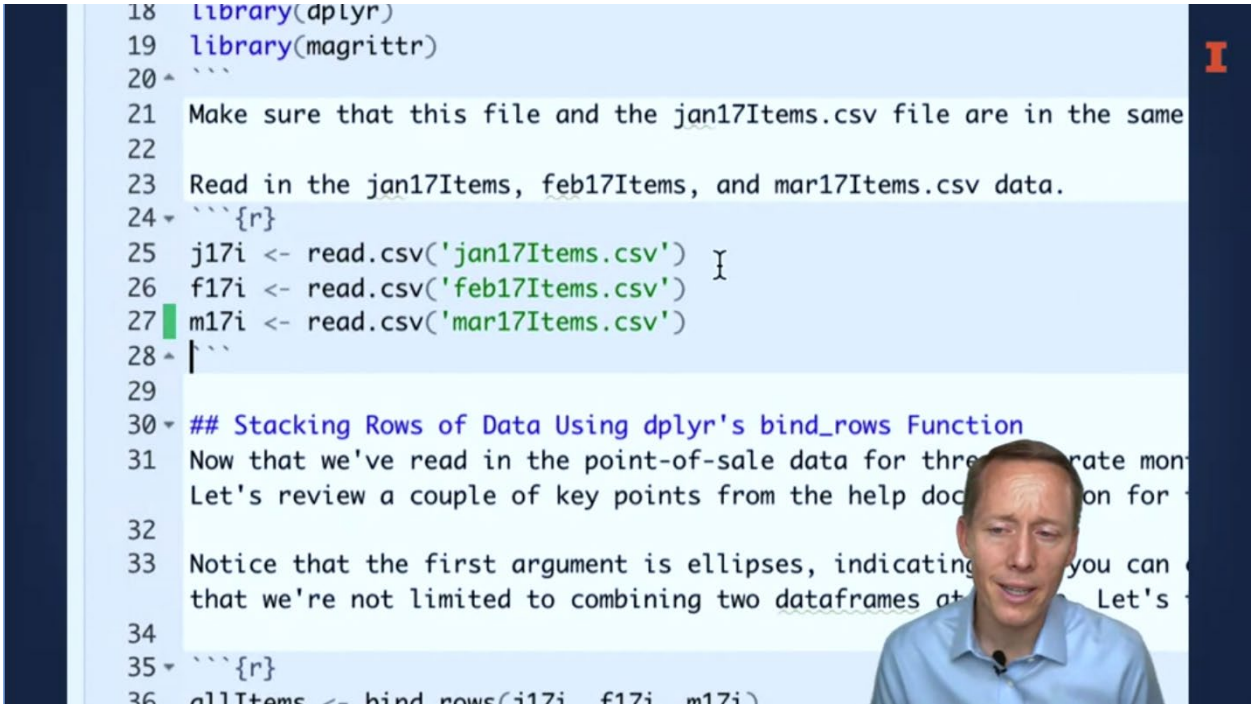
Oftentimes large data sets are broken apart and stored in separate files. For instance, you may have a file for each month's worth of transaction data. Or, if you're pulling data from a large database, you may pull it a day at a time or a week at a time. In either case, it's desirable to stack all the data together so that you have one long data set. This may be known as a vertical stack. There are also horizontal stacks in which you want to stack columns from data frames together and make one very wide data frame. So in this lesson, I want to illustrate how to use the bind rows and bind cols functions from the dplyr package. And also use the arranged function to then sort the data.



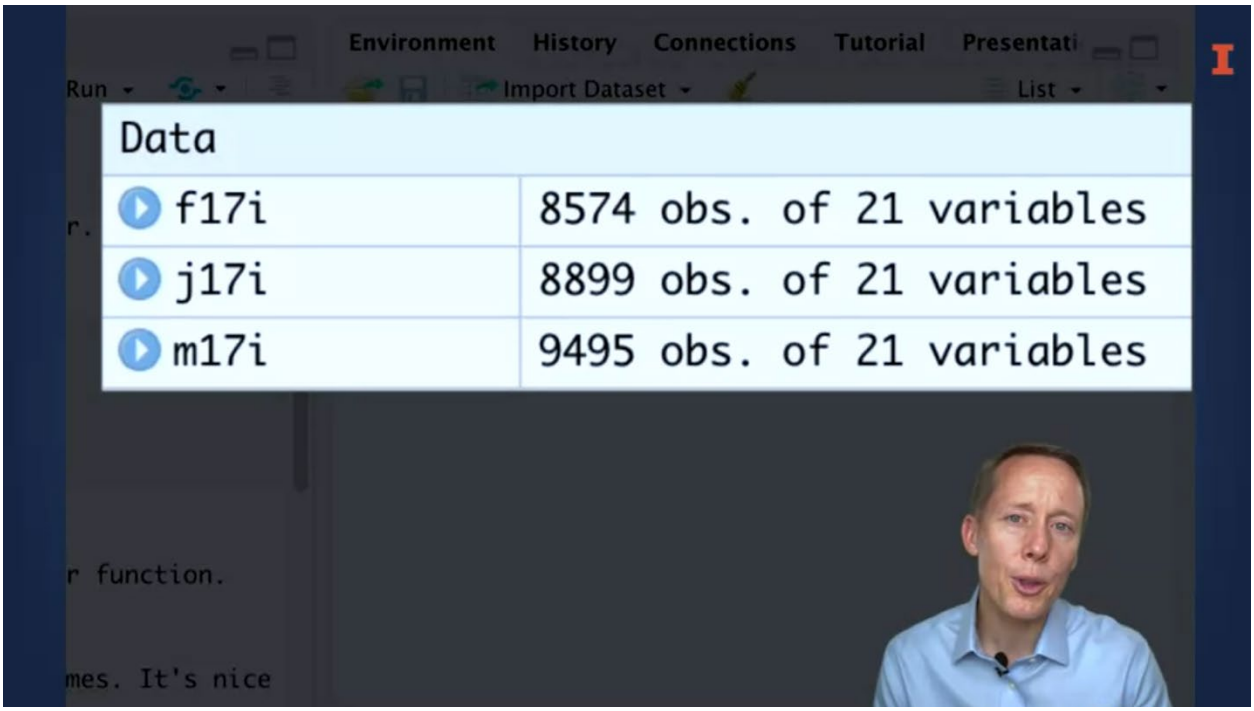
The first thing is that we should make sure that we have loaded the dplyr and magrittr packages.



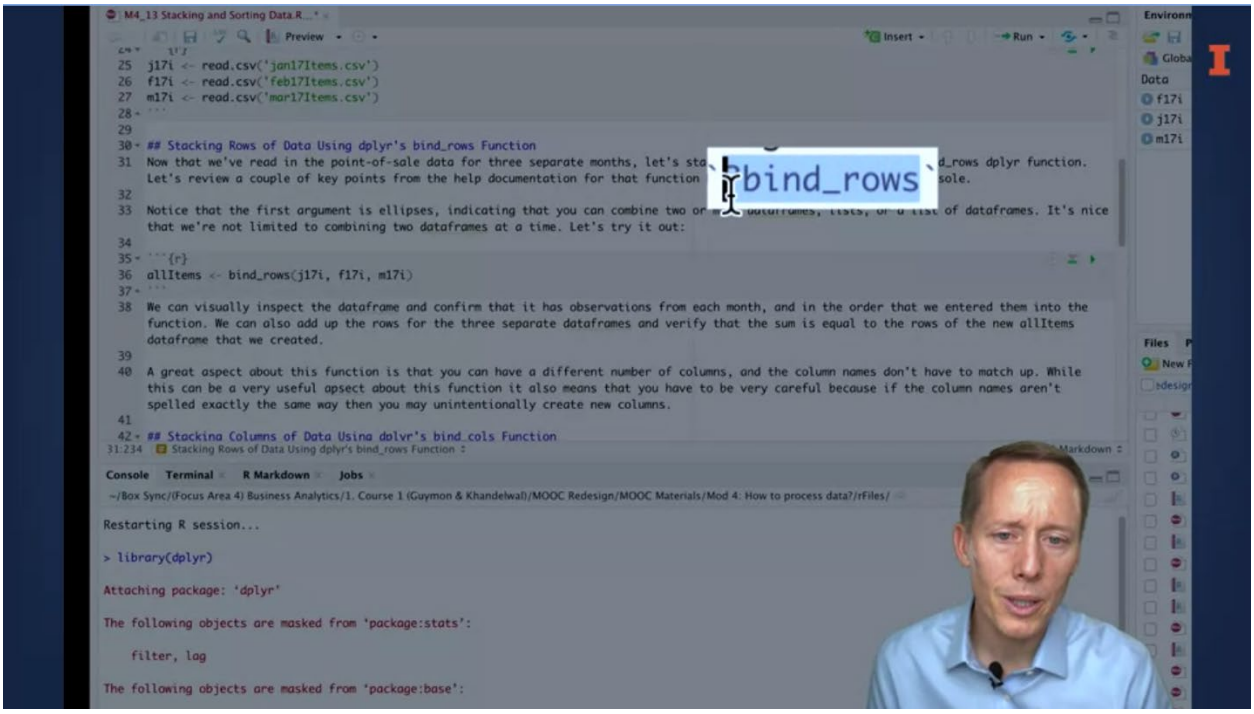
Then make sure that this our notebook file along with the jan 17 items dot CSV file, the feb 17 items dot CSV file and the mar 17 items dot CSV file are all in the same folder. And that the working directory is pointed to that folder.



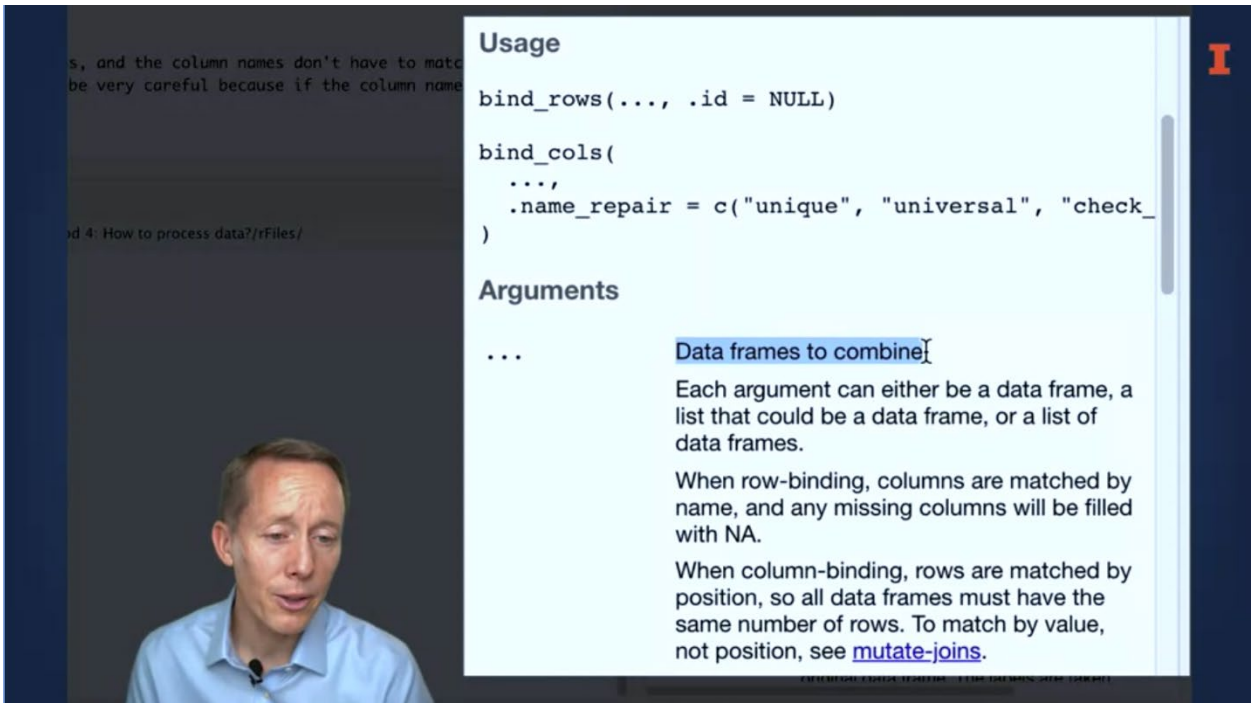
I'm going to go ahead and then read in each of those data sets as j17i, f17i, m17i.



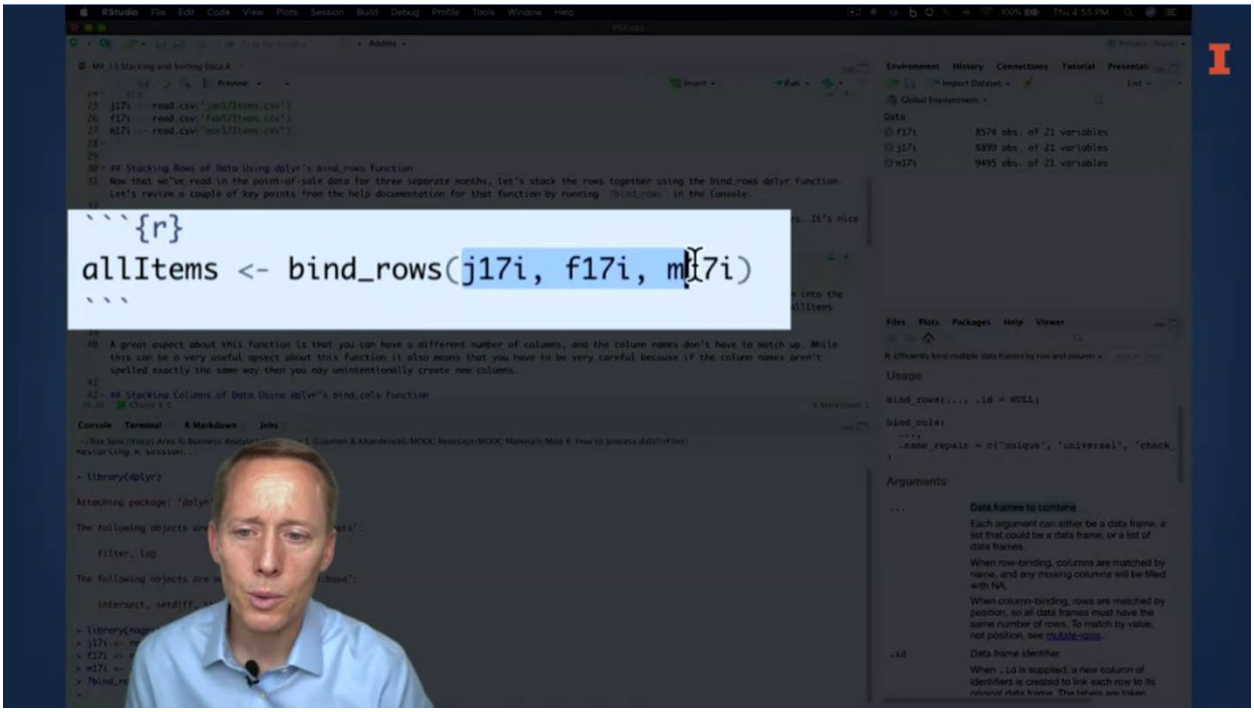
All right, so we essentially have three months worth of transaction level data. If we want to analyze all this data at once, we need to put it into a single data frame. And so we want to bind the rows of data together.



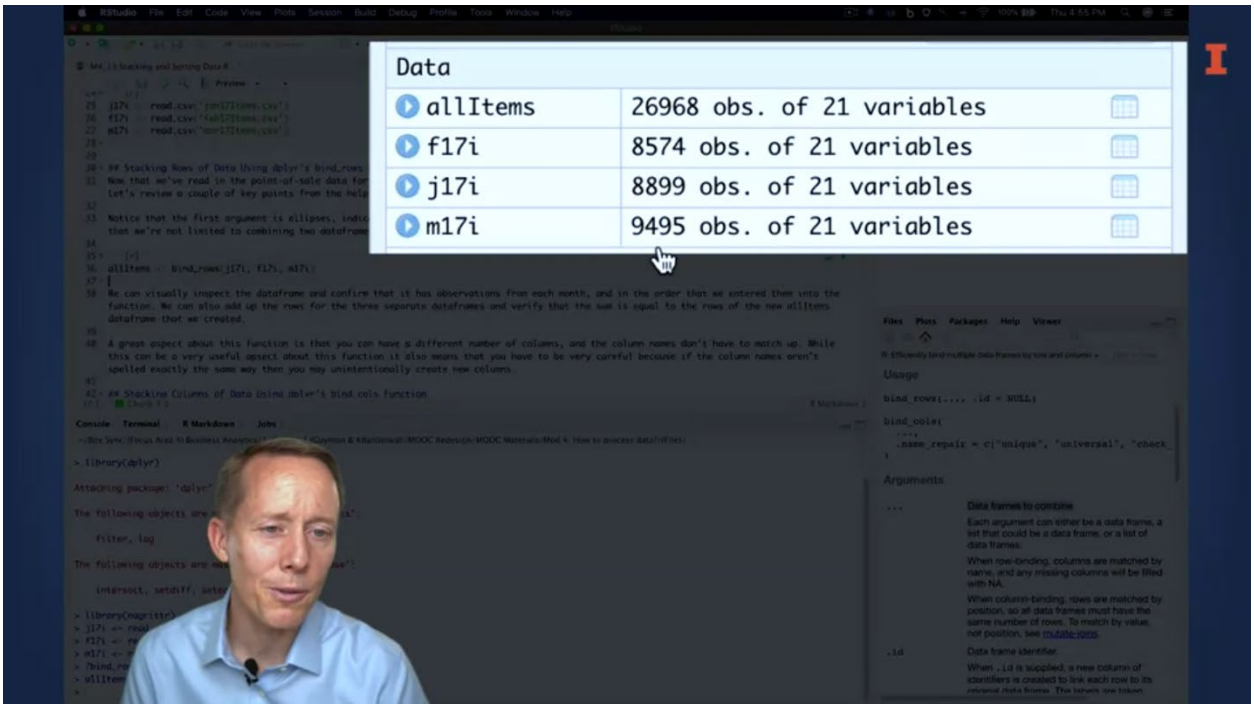
We can do that using this bind rows function. And let's go ahead and read the help documentation for it.



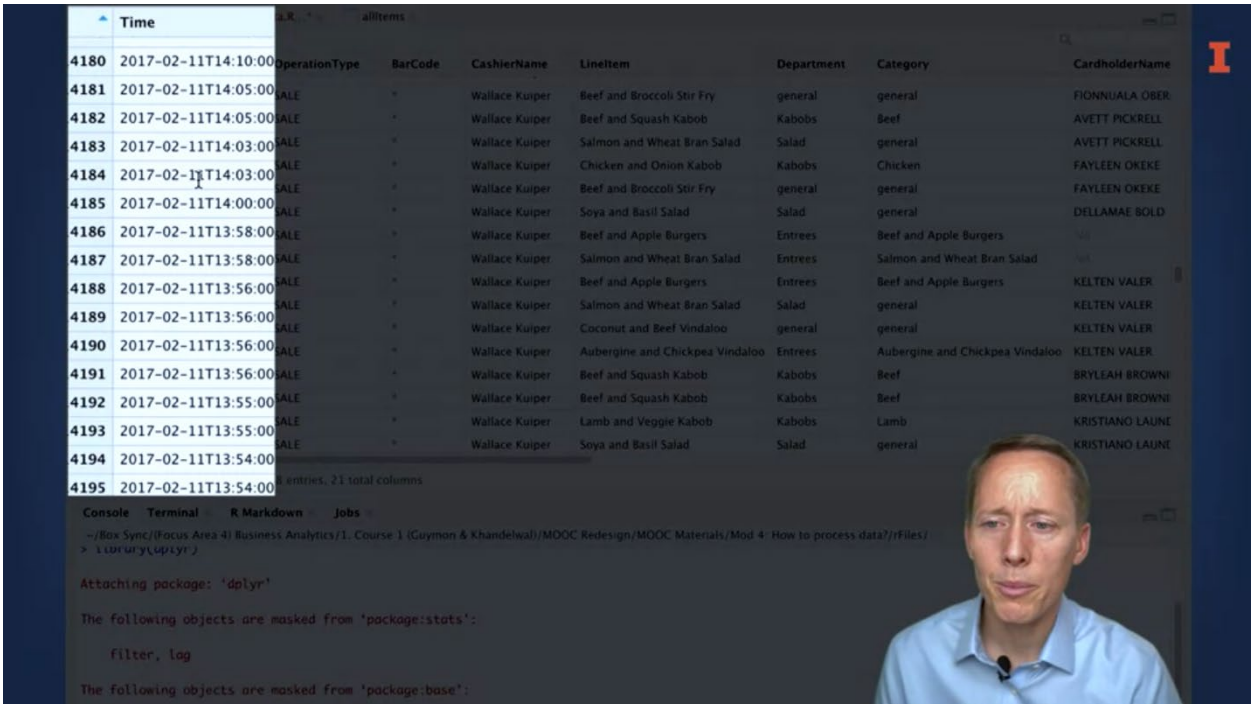
If we go to the usage, it says, bind rows. And the way it's used is there are three ellipses there. We can read more about that. And those ellipses represent the data frames to combine.



So let's go ahead and try that out. We will create a new data frame, allItems. And that will be the result of using the bind rows function. And then we'll enter in their separated by commas j17i, f17i and m17i.

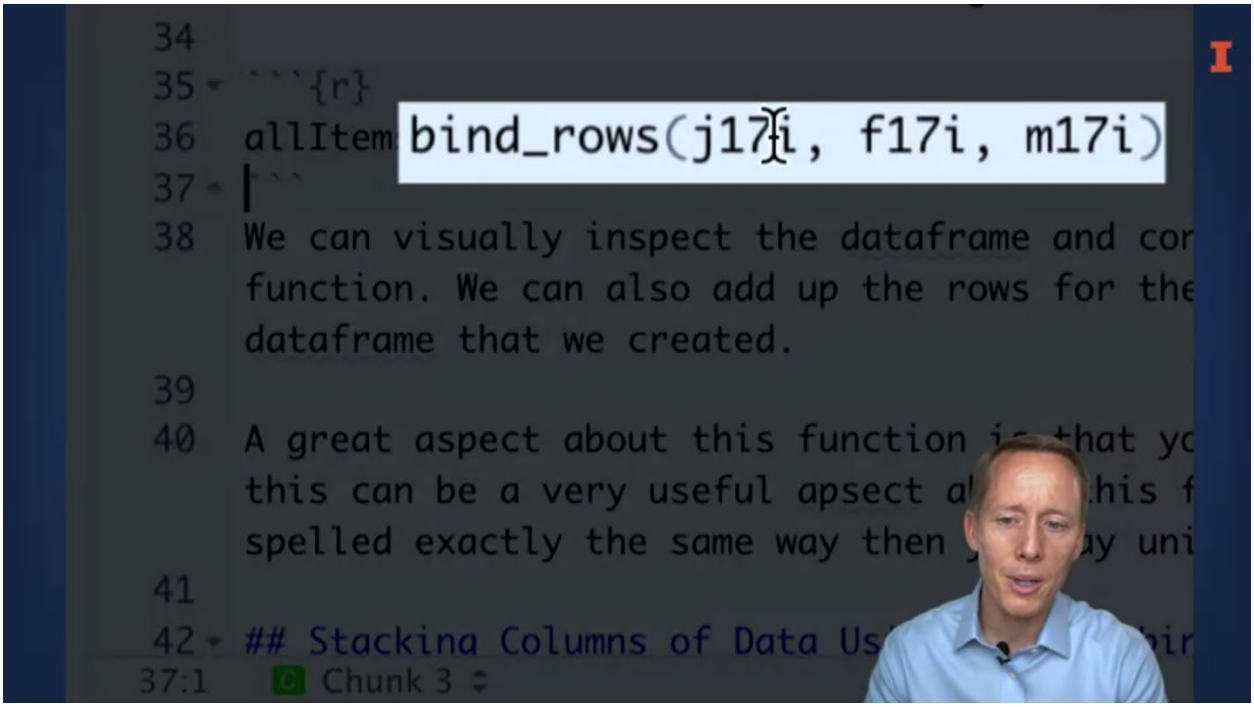


If we run that, we can see this new data frame object, allItems. And we could do a test to make sure what right and what not right. But you could add up the number of observations. And make sure it equals the total and the allItems data frame.

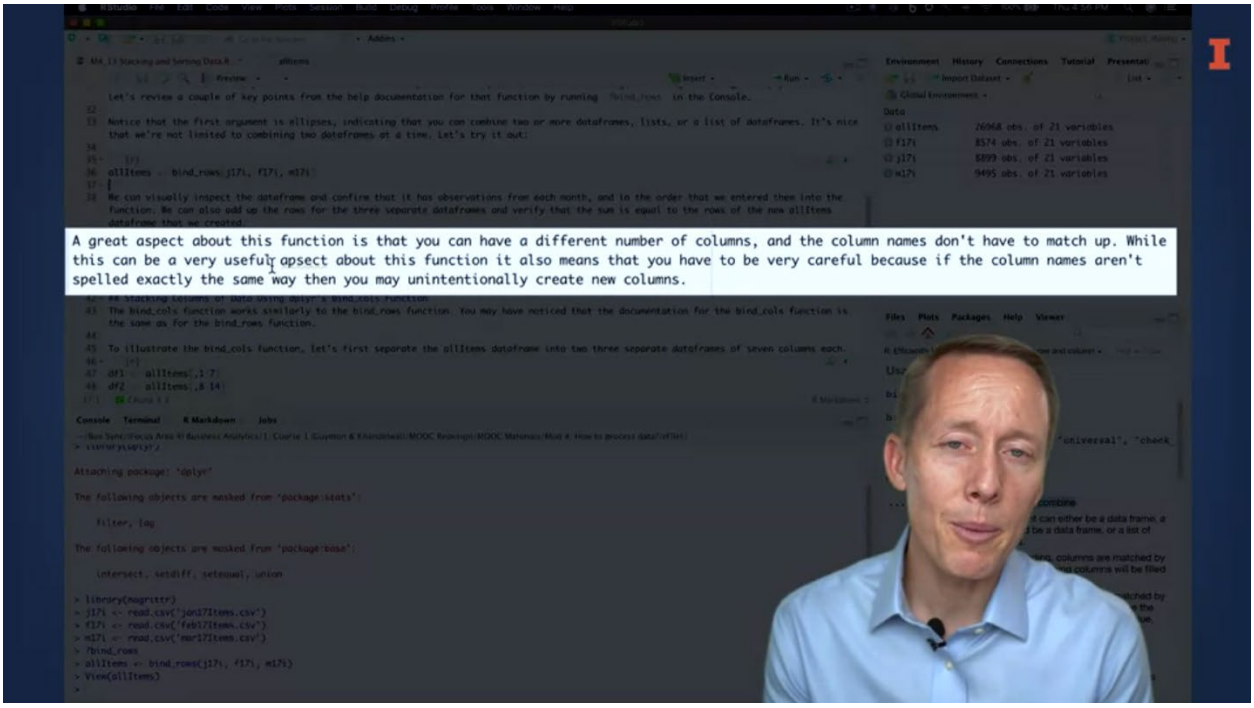


We can also visually inspect this data frame. And I just want to point out one thing which is the order matters. So you can see that the first observation comes from January. If

we go down to about the middle, we're now in the February. And then at the end, we've got the March observations.

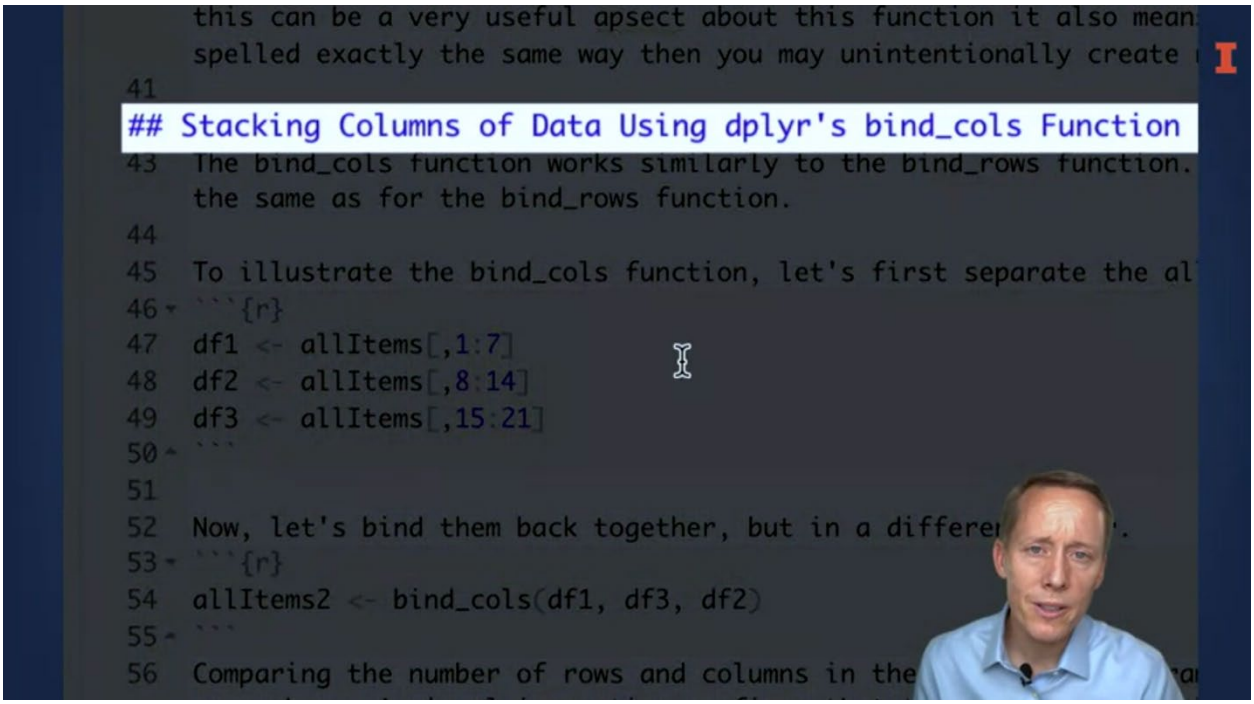


Also, it's basically stacking January on top in February under that and March under that.

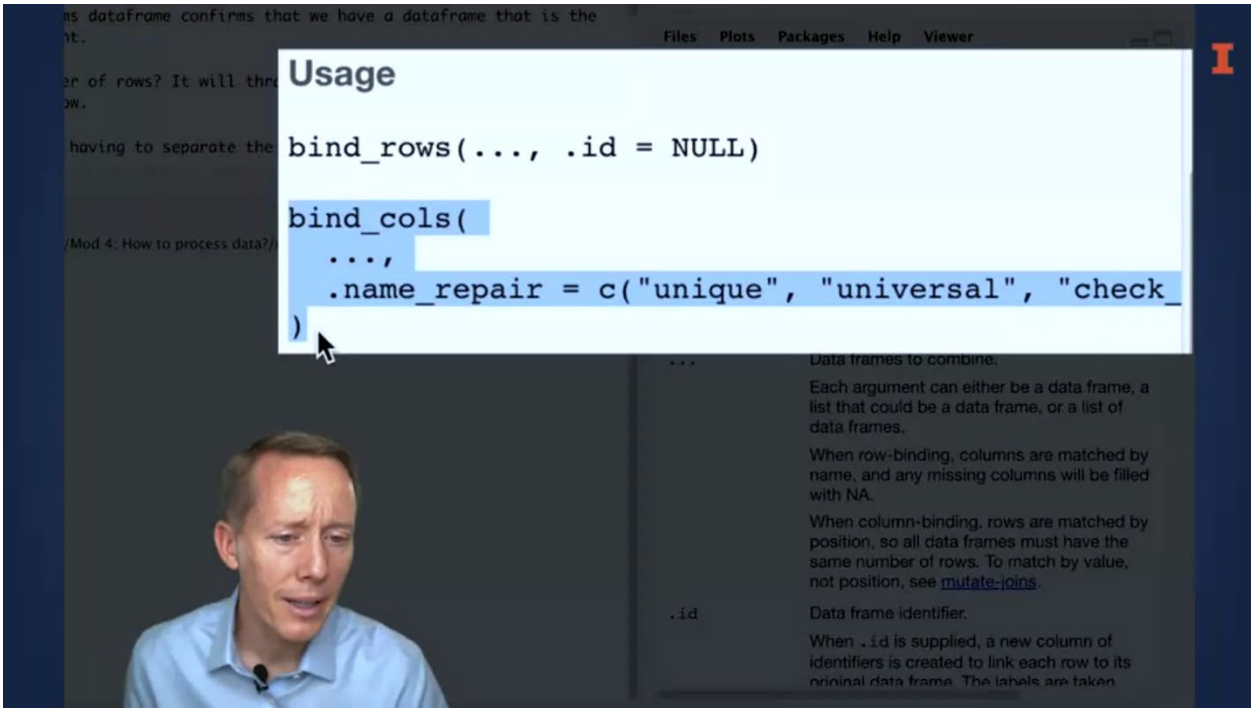


So a great aspect about this function is that if the column names are not the same or the same columns don't exist in each data frame, that's okay. I'll still stack them

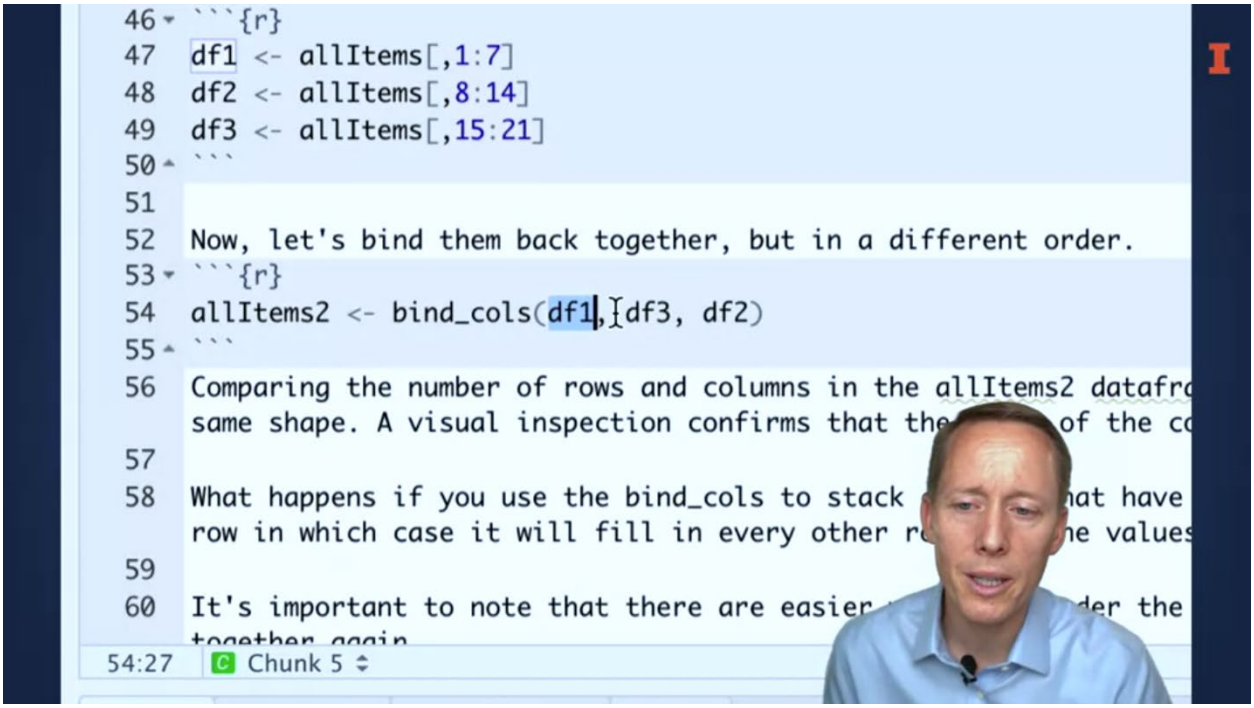
together. And if they're missing from one day to frame, it will just insert an a in there. So that can also be problematic. Because if you have two columns that are essentially the same but one has a capital first letter and the other doesn't, it won't know the difference. And it will create two separate columns. So in either case, you need to be careful to make sure that once you've blinded the rows together that you've got the data frame that you expected to get.



All right, let's also talk briefly about binding columns of a data frame together. I don't do this as often. But it can be done using the bind cols function.

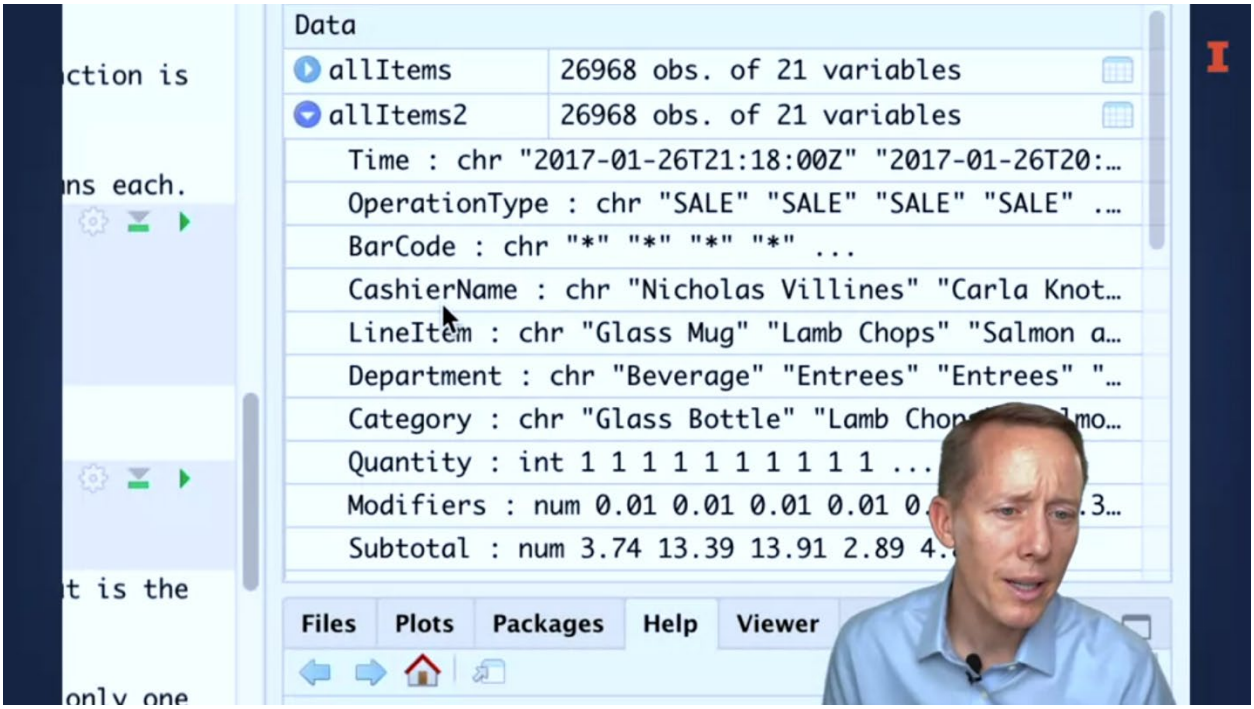


You may have noticed in the help documentation for bind rows, we've got bind cols. And it's very similar to how bind rows works.

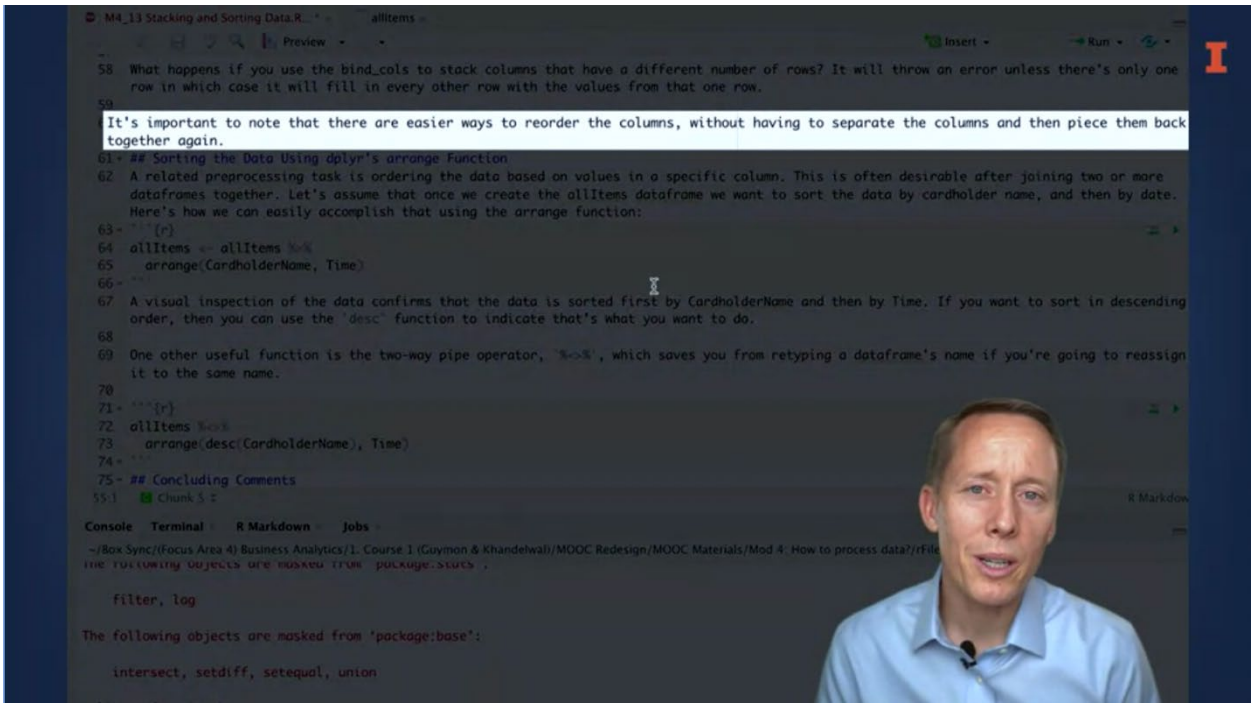


For illustration purposes, we're going to take the observations in allItems data frame and split those out into three separate data frames, such that df1 has columns 1 through 7. Df2 has columns 8 through 14. And df3 has columns 15 through 21. All right, and then we'll bind them back together into allItems2. But just to see how it works, we're going to

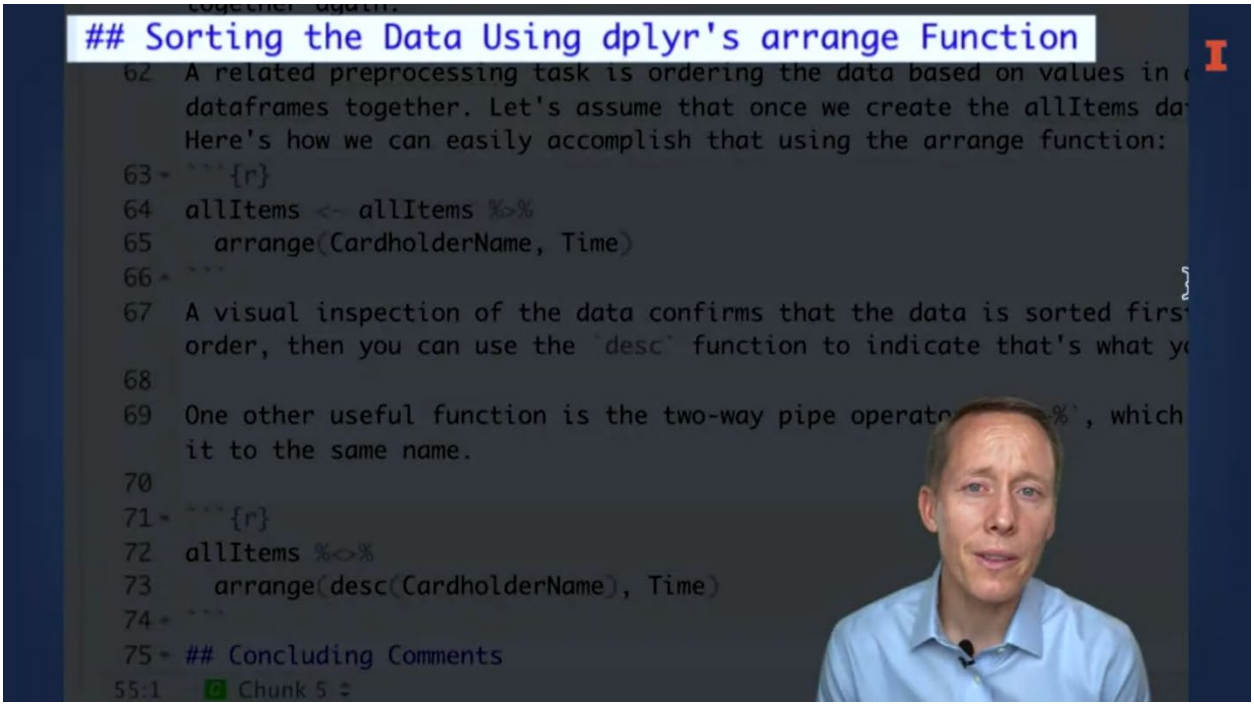
bind them together in a different order. So we'll start with df1, then df3 and then df2. So go ahead and run that.



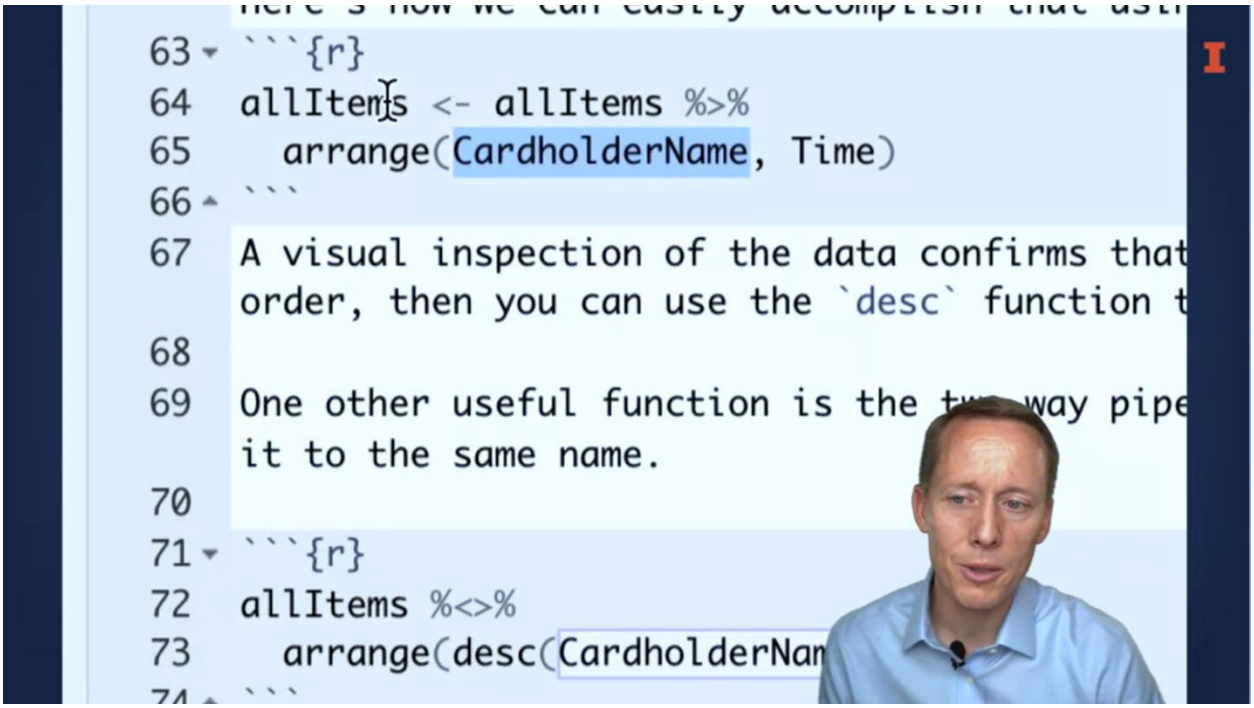
And if we look at the column names for allItems2, it starts with time, it goes down to category and then quantity. All right, if we compare that to allItems that goes from time down to category and then cardholder names. So again, the order matters in which you enter these data frames into the bind cols function.



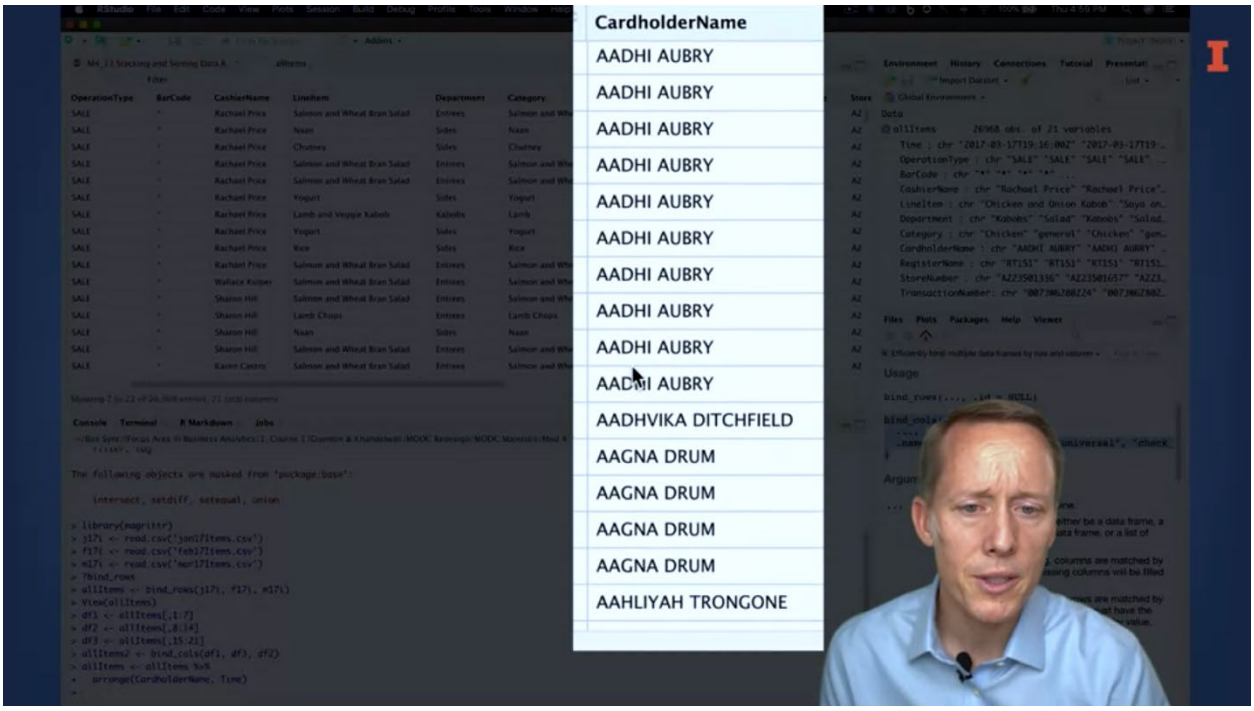
Now it's important here to recognize that there are easier ways to rearrange the order of columns in the data frame. You don't have to separate them out and then bind them together again in the order that you want. But that's for a separate lesson.



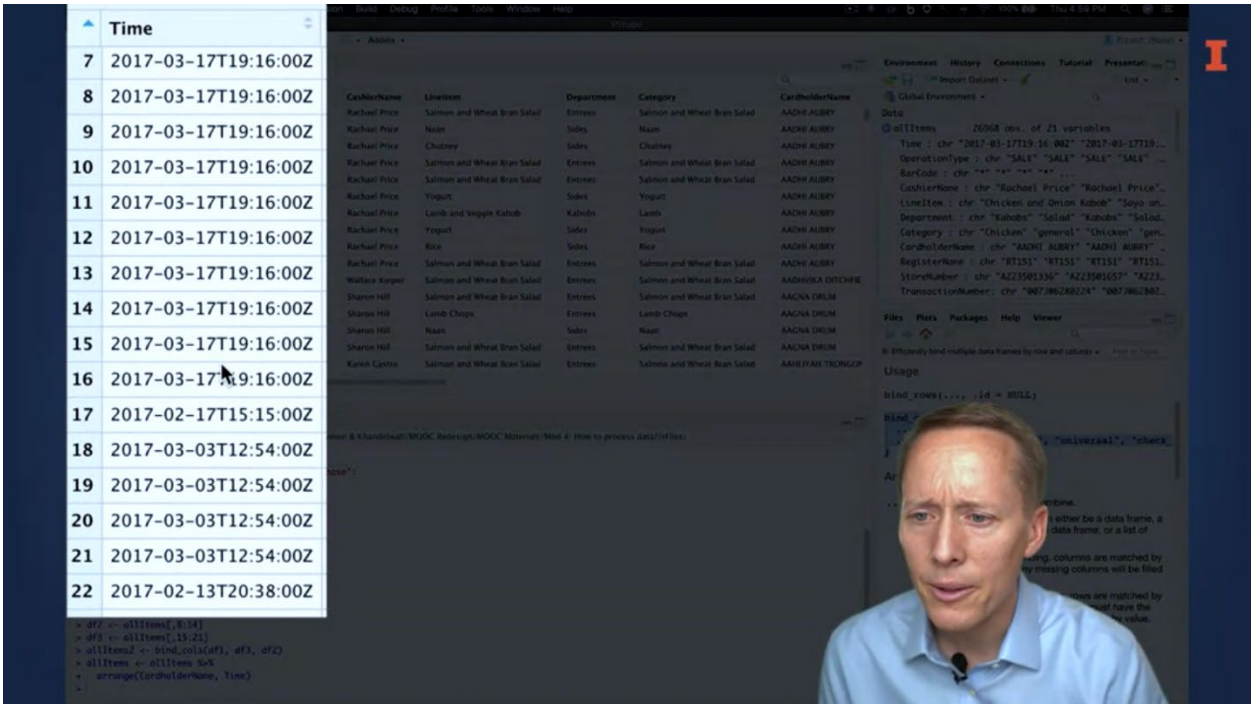
Now, once you've bound data frames together, often it's the case that you want to arrange the observations in a certain way. And we can do that using the arrange function from the dplyr package.



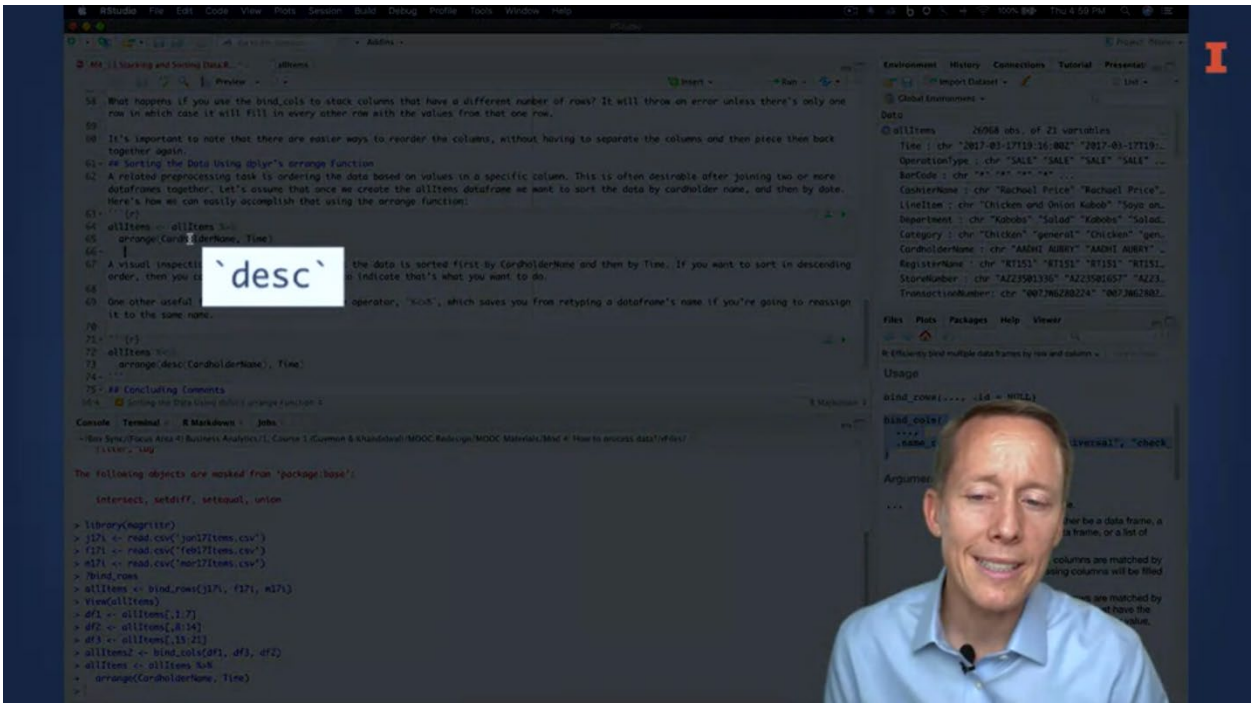
So I want to illustrate that using this allItems data frame. And we'll take that and we'll pipe it into the arranged function. And it's very simple to use. All we will do is enter the column names by which we want to sort the data. And then feed it into this back into the allItems data frame object. So in this case, what we're doing is we're going to say we're going to sort the data first by cardholder name. And then we're going to sort it by time. So that the earliest transactions for each cardholder comes first. So let's go ahead and do that.



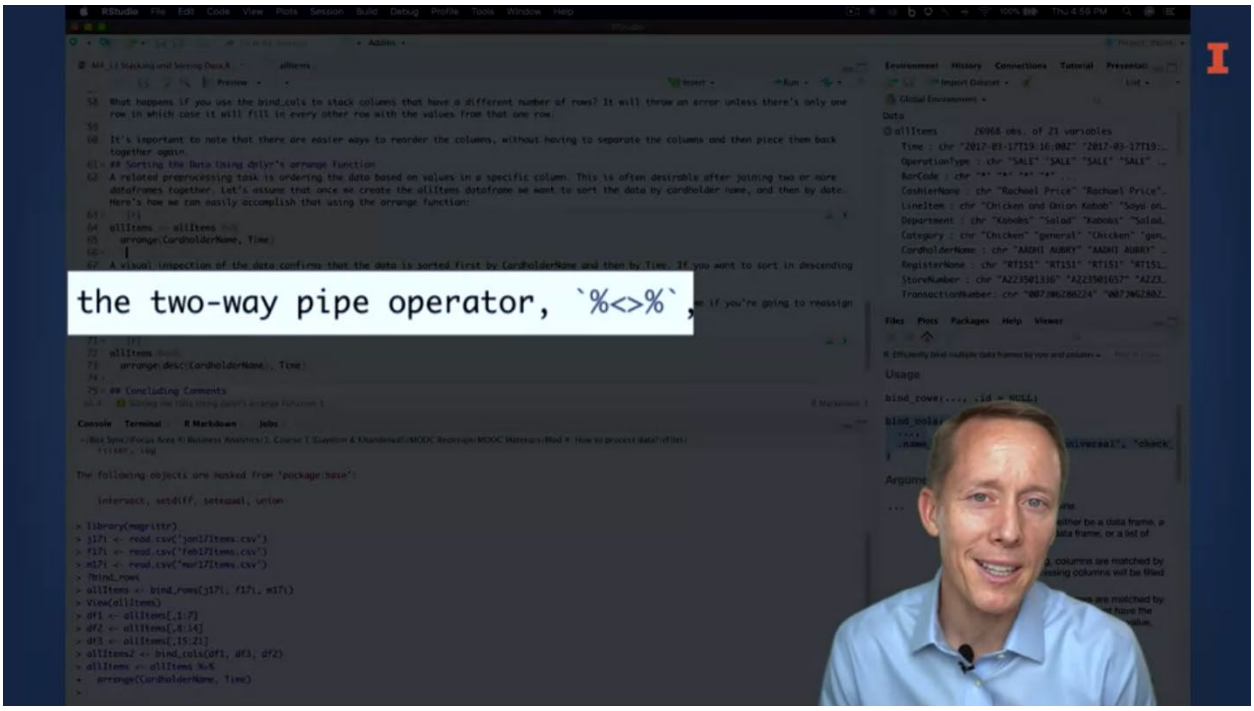
Let's just visually inspect the data here. If we look at cardholder name, we can see that the first 16 or so observations are all from the same cardholder, AADHI AUBRY.



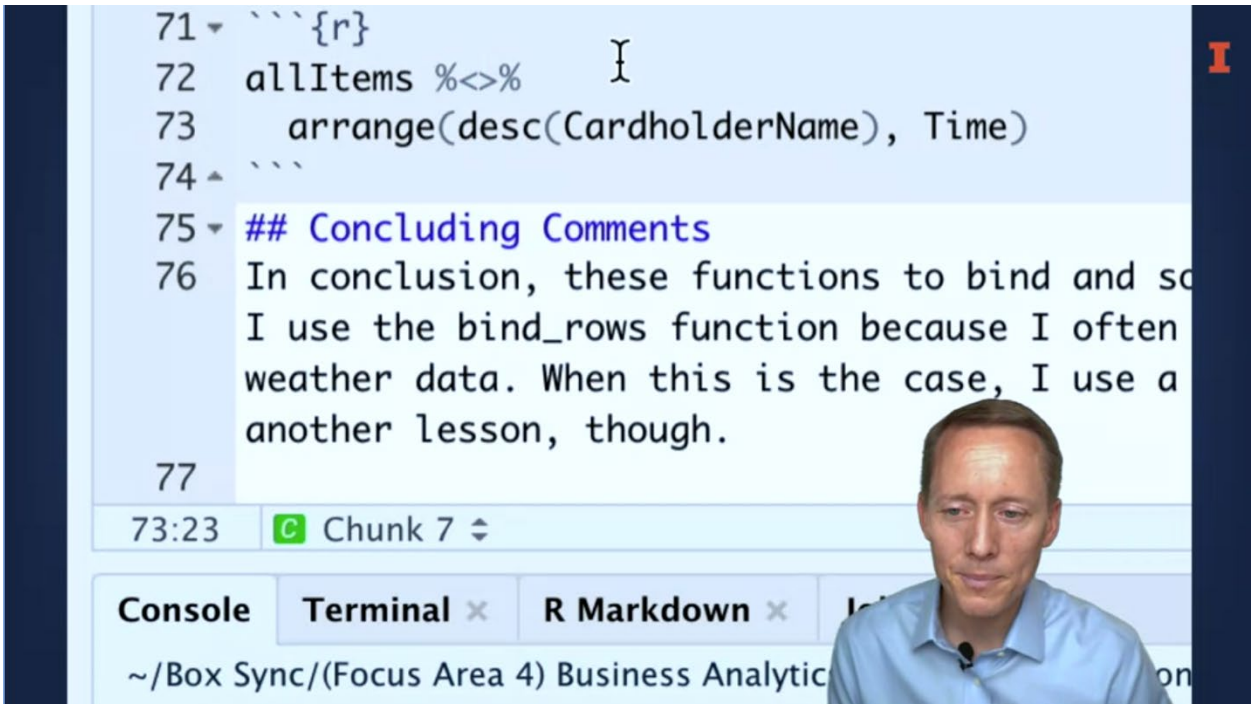
And if we look at the time in this case, they're all the same date. Yeah, they're all the same date, so we can't really see. But it is sorted by time as well. All right, so that's how you can sort data very quickly. It's very easy to do.



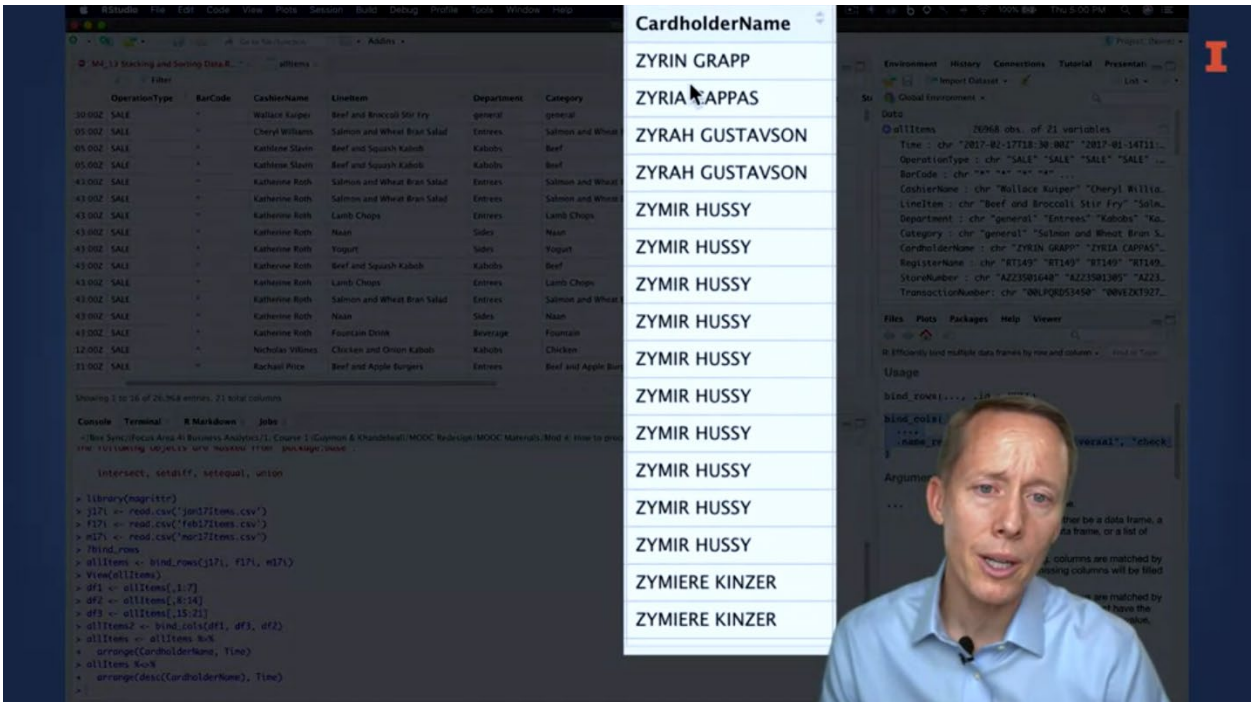
Now, you can also sort in descending order. And you can do that by using the dsc function which stands for a descend.



So to illustrate this, I also want to introduce you to a two-way pipe operator. It's very easy to use. Okay, and the way this operator works is very similar to the regular pipe operator. But before the greater than sign, we're going to enter a less than sign. And you can think of these as arrows.



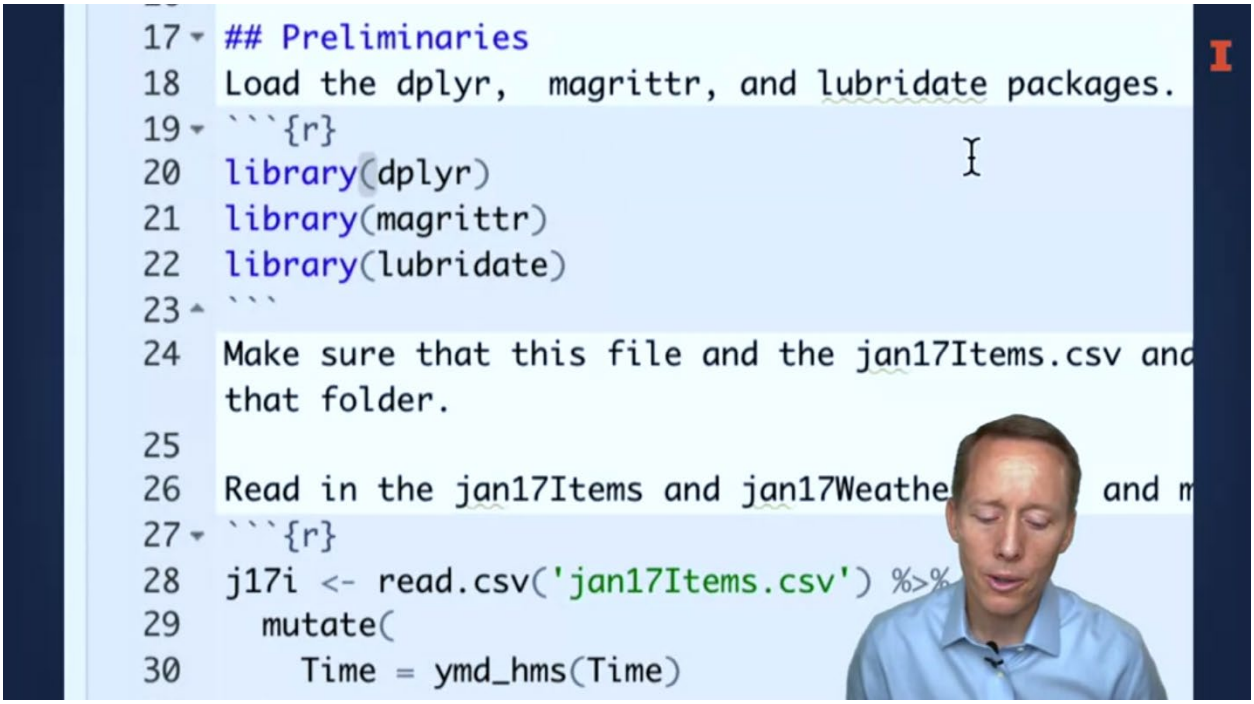
All right, so let's look at the code for how this works. What we're doing here is we're going to take the allItems data frame and then we've got this two-way pipe operator. And at this point, focus on the greater than signs. So we're feeding allItems, were piping it into the arrange function. And then we're going to sort in descending order by cardholder name. And then in ascending order by time. And then what we're doing is, if you focus on this less than sign, we're piping that back into the allItems data frame. So the purpose of this two-way pipe operator is to reduce the number of times you have to write out the same data frame.



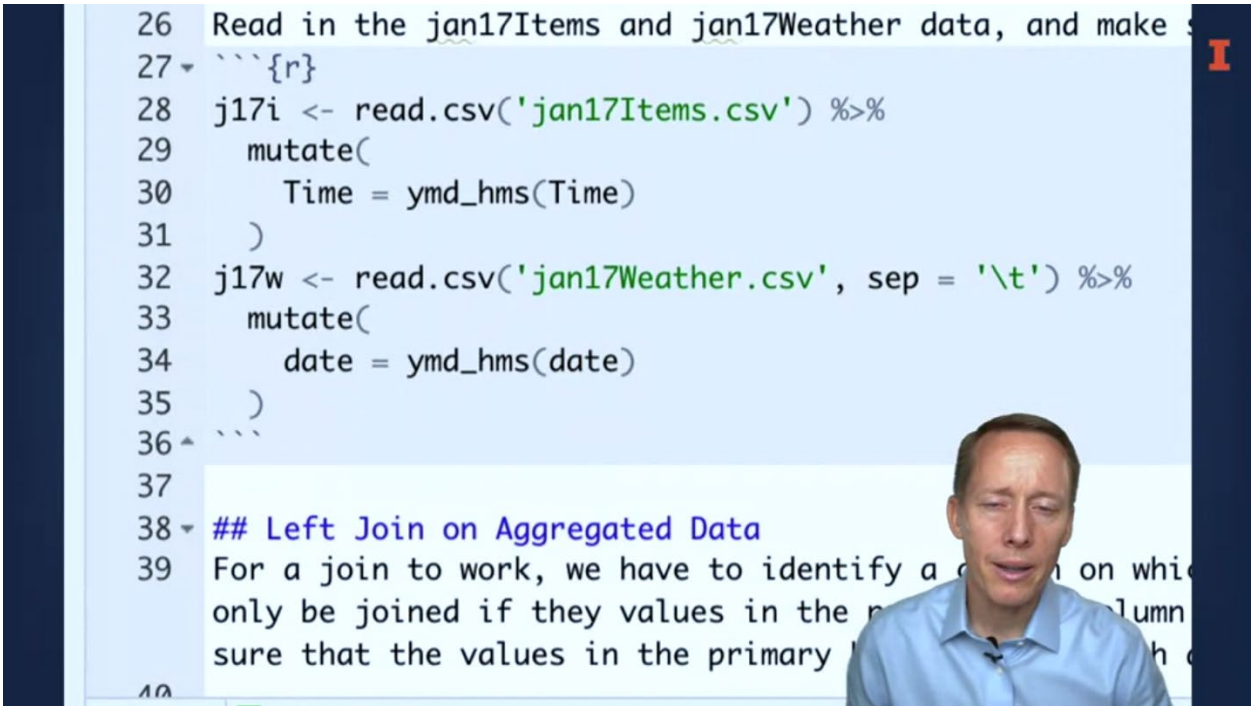
All right, so I'll go ahead and run this. And let's visually inspect the allItems data frame and go to cardholder names. Sure enough, now you can see it sorted in descending order by cardholder name. because ZYRIN GRAPP, it starts with a Z. So now, I mentioned that I don't use the bind cols very often. If I want to combine data frames that, like weather data to transaction level data, bind cols is not the correct way to do that. We would want to join data together. That's a topic for another lesson. At this point though, I hope you recognize that it's very easy to stack data frames together and sort them.

Lesson 4-1.14 Joining Data

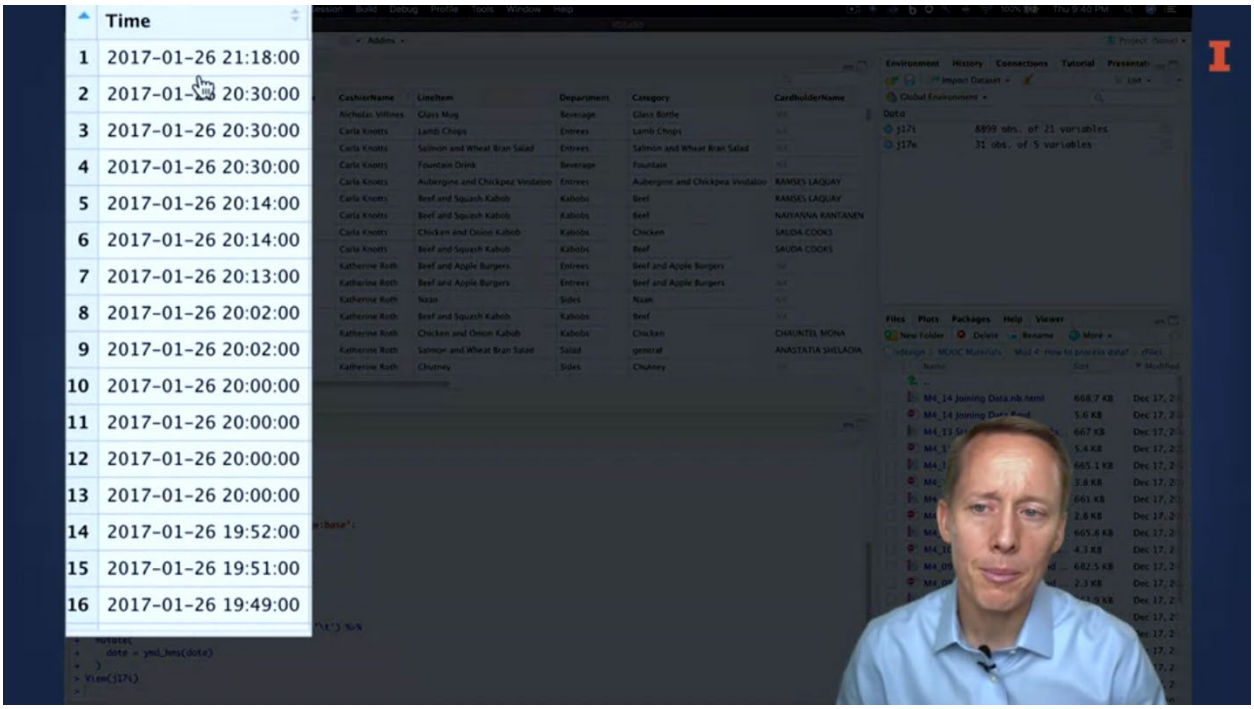
Oftentimes useful insights can be gained by combining two data sets together. For instance, in this lesson, we're going to combine weather data with point of sale data, so that we can evaluate the influence of weather patterns on customer behavior. When you combine data sets together, you often do this using joint functions. So I'll illustrate how to use several joint functions in this lesson.



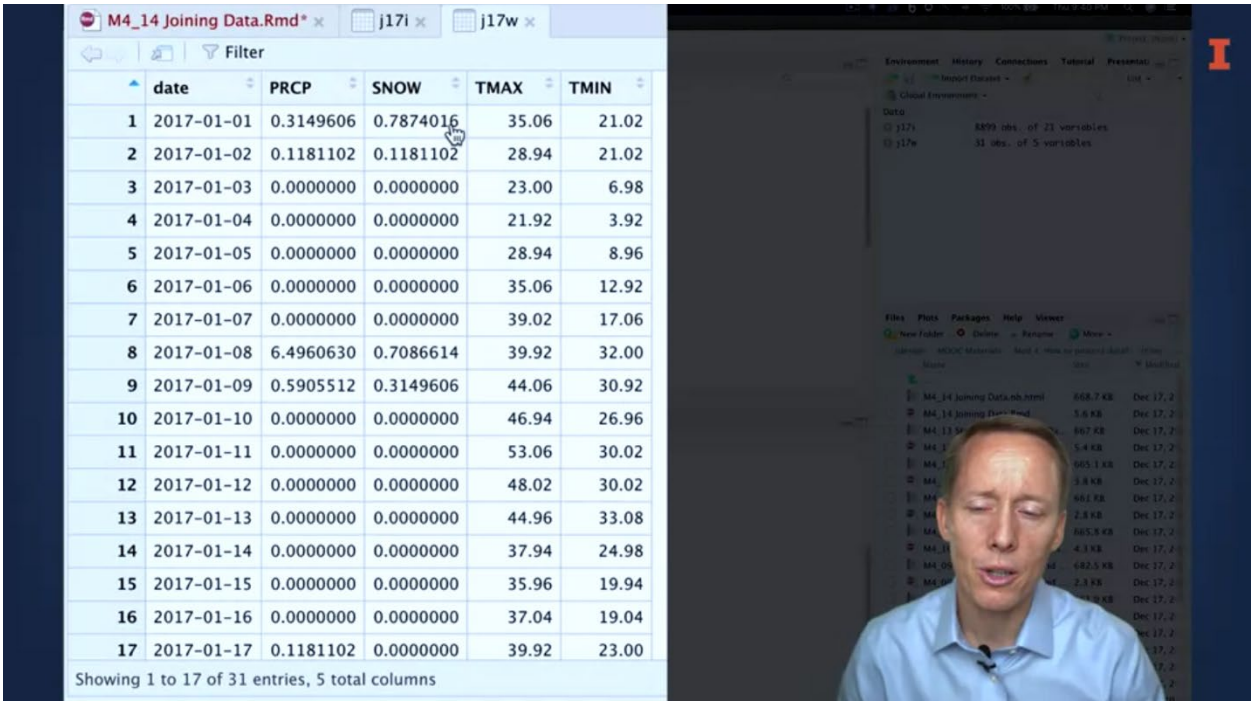
First, I'm going to make sure that I have installed and loaded the dplyr magrittr and lubricate packages.



Next, I'm going to read in a point of sale data file the jan17Items data file. As well as the jan17Weather data file, which contains weather data. So I'll create the j17i data frame object by reading in the jan17Items.csv file. And converting the Time column from a character string format to a date Time format using the ymd_hms function. Next, I'll create the j17w data frame object by reading in the jan17Weather.csv file. And converting the date column to a date time format using again the ymd_hms function.

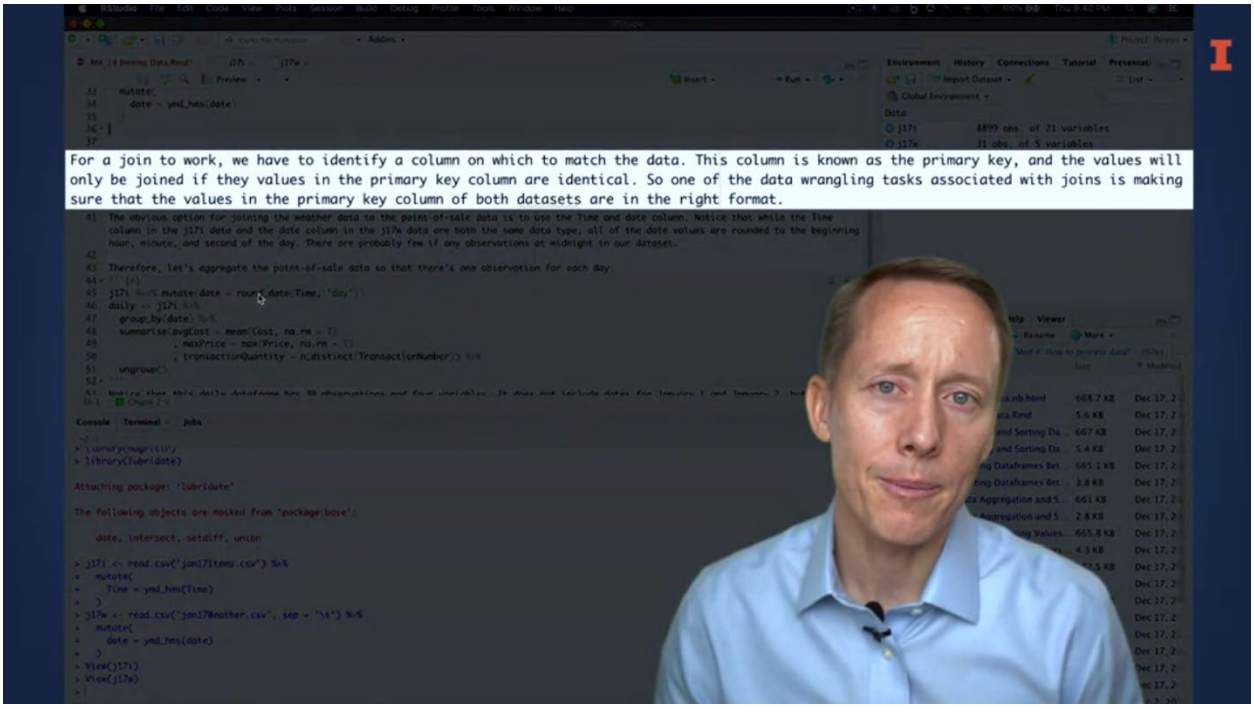


Now, let's go ahead and just quickly explore these two data frames j17i, the first column is time there, and it is now a date time format.

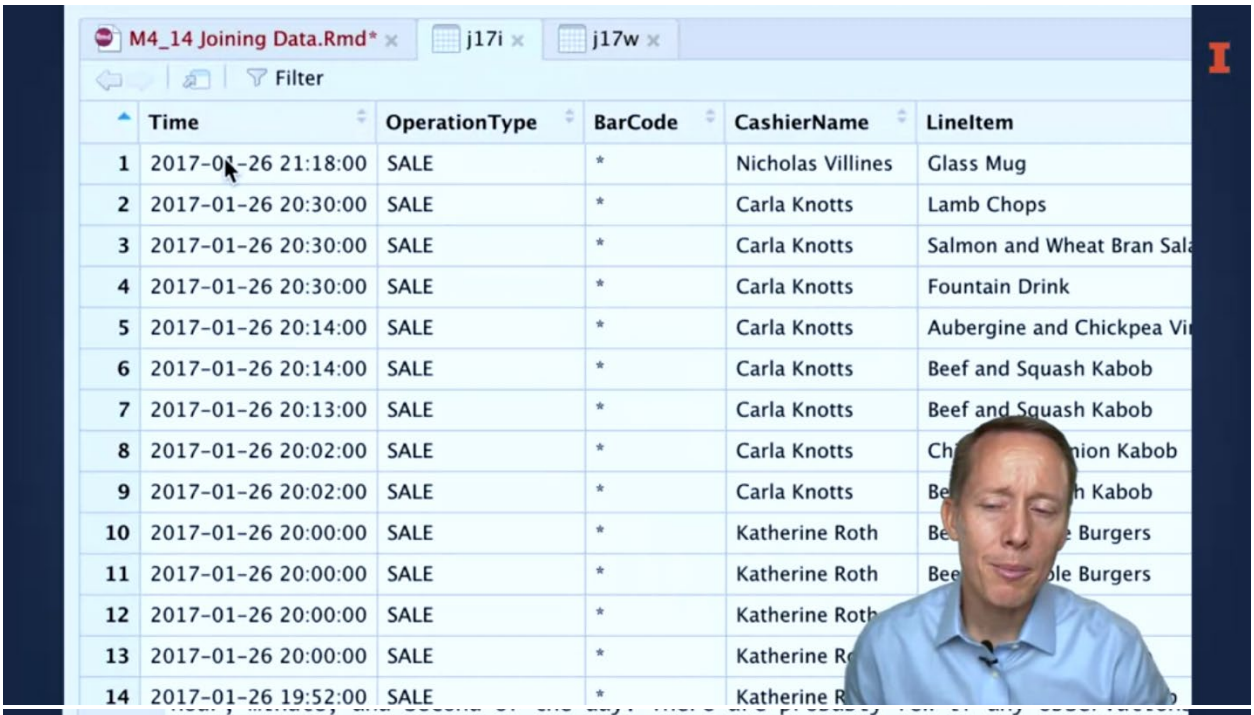


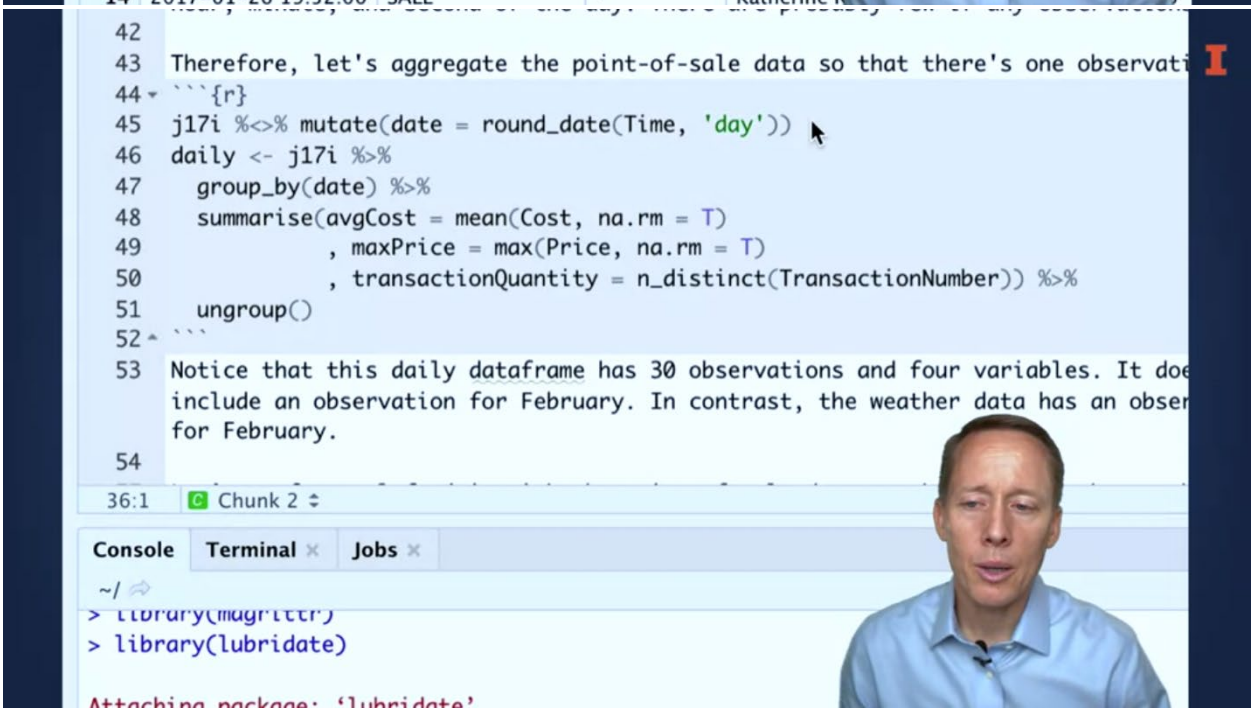
We've got a bunch of other columns. The weather data only has five columns in it. We've got the first column, which is a date column. And then we've got PRCP SNOW and TMAX and TMIN temperature. And real quickly, you can see we've got one

observation for each day. At this point, we need to think about how to join the weather data with the point of sale data.

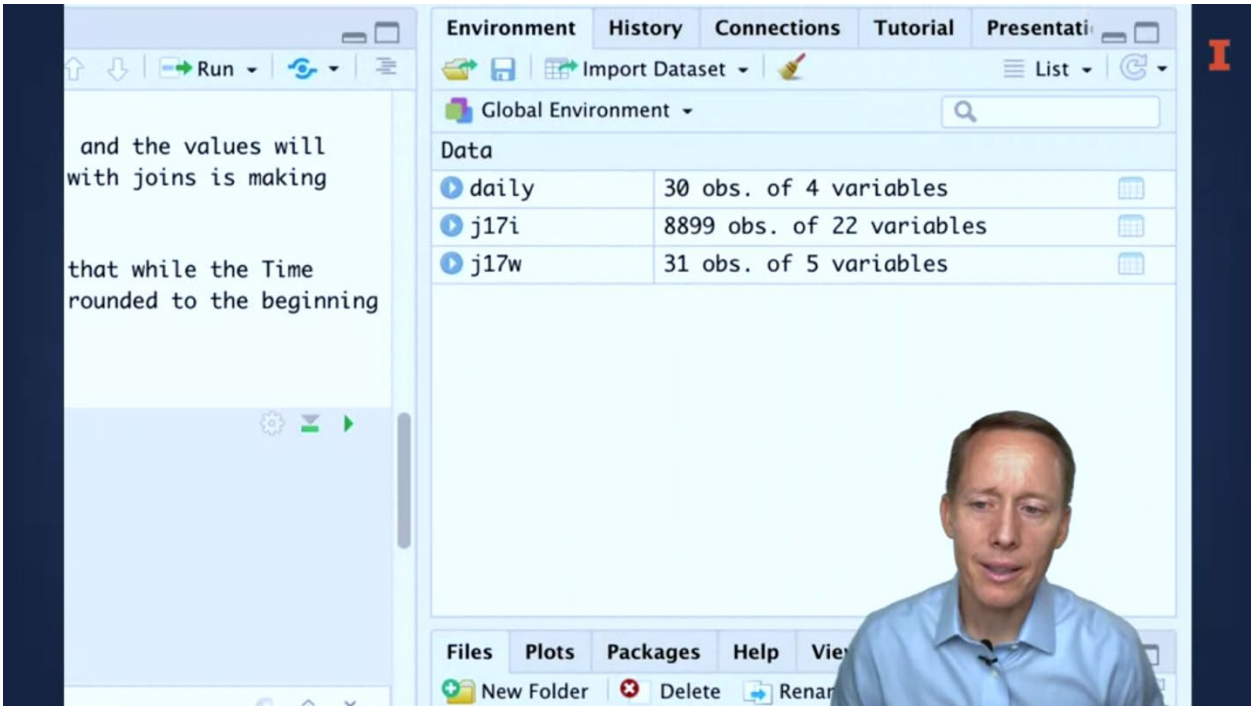


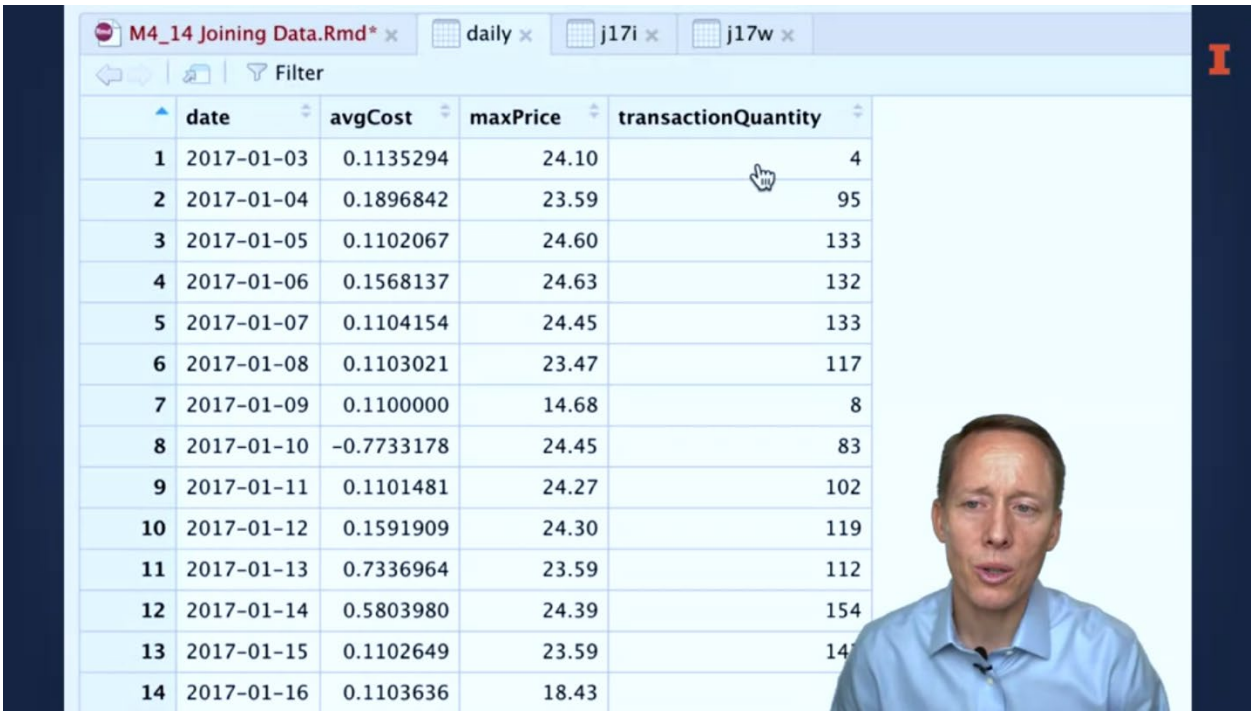
Now, whenever you join data together, the main thing is that you have to find a column in both data sets on which you can match. And this is known as the primary key column. And it's critical that the values in the primary key column are identical so we can match it up.





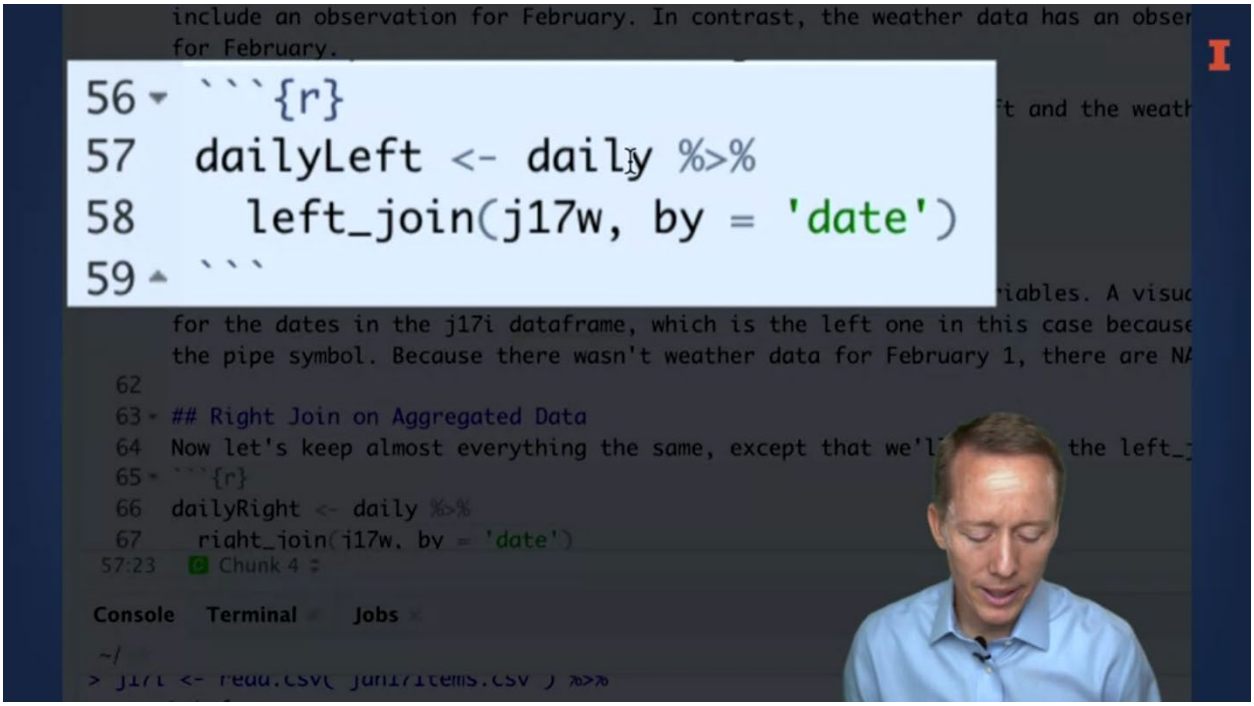
Now the obvious option in these two data frames is to join on the date and time columns. However, the Time column is not rounded to the nearest day. So the first step that we need to do is convert the Time column to a day column or just a date. So I will do that with this step right here. I'm going to take the j17i data frame object, and pipe it into the mutate function and create a new column called date. And I'll use around date function to round the time value to the nearest day. Now, the next thing I'm going to do for simplicity sake is to. I create an aggregated data set for joining.





	date	avgCost	maxPrice	transactionQuantity
1	2017-01-03	0.1135294	24.10	4
2	2017-01-04	0.1896842	23.59	95
3	2017-01-05	0.1102067	24.60	133
4	2017-01-06	0.1568137	24.63	132
5	2017-01-07	0.1104154	24.45	133
6	2017-01-08	0.1103021	23.47	117
7	2017-01-09	0.1100000	14.68	8
8	2017-01-10	-0.7733178	24.45	83
9	2017-01-11	0.1101481	24.27	102
10	2017-01-12	0.1591909	24.30	119
11	2017-01-13	0.7336964	23.59	112
12	2017-01-14	0.5803980	24.39	154
13	2017-01-15	0.1102649	23.59	14
14	2017-01-16	0.1103636	18.43	

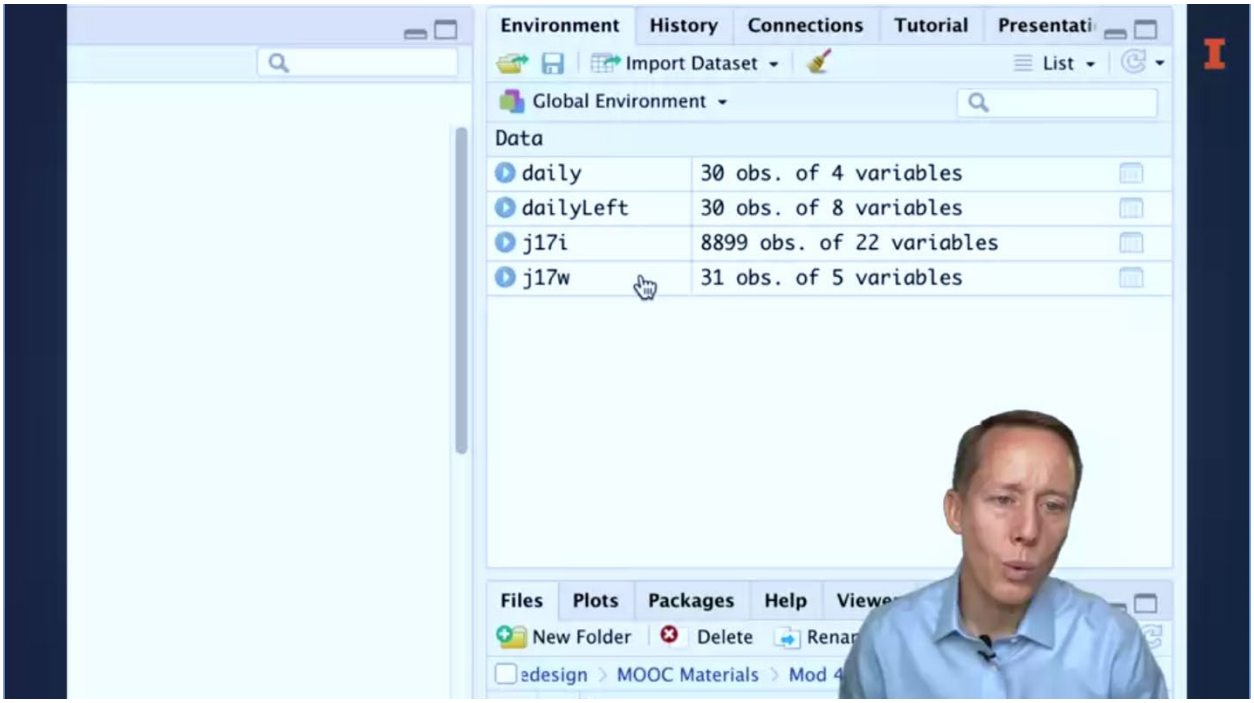
And I will do that by creating this daily data frame. And as you look at this daily data frame, you can see we've got one observation for each day. And then we've got the average cost, the maximum price and the total transaction quantity, the total number of transactions.



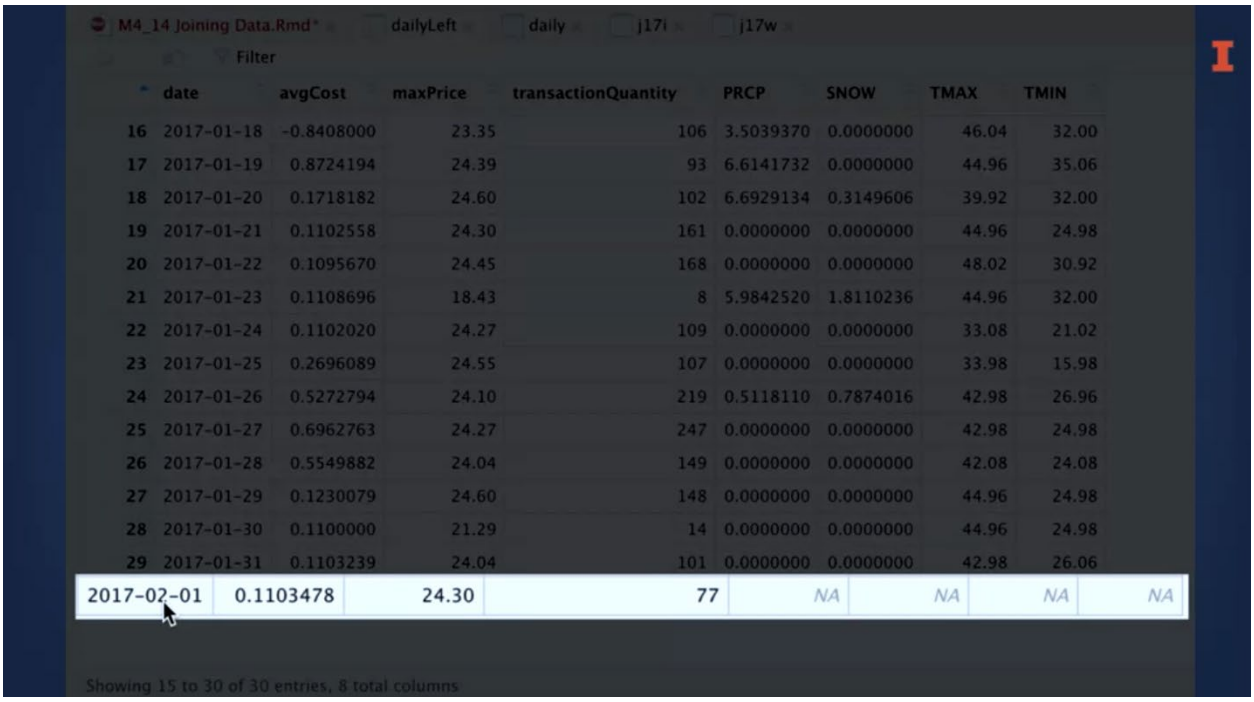
So at this point, we're ready to perform some joins. So let's start by illustrating how to use a left join. So the way I'm going to do that is I'll create a new data frame object dailyleft. And I will start by taking the daily data frame, and I will pipe that into the left join function, and then I will. The next argument is the data frame that I will be joining columns from. So that's the weather data frame, and I'm going to join them based on this primary key column, which is date. So let's go ahead and do that.

	date	avgCost	maxPrice	transactionQuantity	PRCP	SNOW	TMAX	TMIN
1	2017-01-03	0.1135294	24.10	4	0.0000000	0.0000000	23.00	6.98
2	2017-01-04	0.1896842	23.59	95	0.0000000	0.0000000	21.92	3.92
3	2017-01-05	0.1102067	24.60	133	0.0000000	0.0000000	28.94	8.96
4	2017-01-06	0.1568137	24.63	132	0.0000000	0.0000000	35.06	12.92
5	2017-01-07	0.1104154	24.45	133	0.0000000	0.0000000	39.02	17.06
6	2017-01-08	0.1103021	23.47	117	6.4960630	0.7086614	39.92	32.00
7	2017-01-09	0.1100000	14.68	8	0.5905512	0.3149606	44.06	30.92
8	2017-01-10	-0.7733178	24.45	83	0.0000000	0.0000000	46.94	26.96
9	2017-01-11	0.1101481	24.27	102	0.0000000	0.0000000	53.06	30.02
10	2017-01-12	0.1591909	24.30	119	0.0000000	0.0000000	48.02	30.02
11	2017-01-13	0.7336964	23.59	112	0.0000000	0.0000000	44.96	
12	2017-01-14	0.5803980	24.39	154	0.0000000	0.0000000	37.94	
13	2017-01-15	0.1102649	23.59	147	0.0000000	0.0000000	35.96	
14	2017-01-16	0.1103636	18.43	18	0.0000000	0.0000000	37.04	
15	2017-01-17	-0.7083983	21.70	82	0.1181102	0.0000000	39.92	
16	2017-01-18	-0.8408000	23.35	106	3.5039370	0.0000000	46.[illegible]	
17	2017-01-19	0.8724194	24.39	93	6.6141732	0.0000000		

And let's just visually explore the daily left data frame. You can see that now we've got in addition to the average cost maxPrice and transactionQuantity. We have the Precipitation Snow. TMAX and TMIN columns as well.

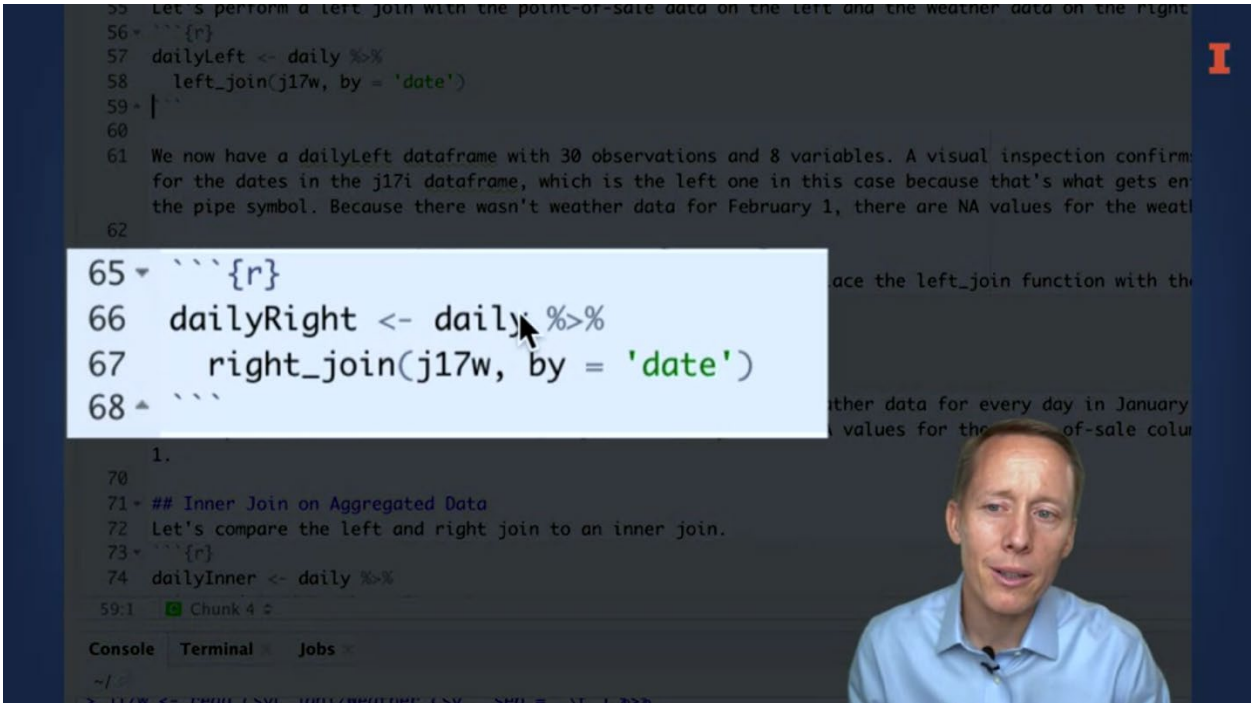


And as we look at the dimensions for this dailyleft data frame has the same number of observations as the daily data frame. Now, this is important because the weather data has 31 observations in him, and has one observation for each day in January.

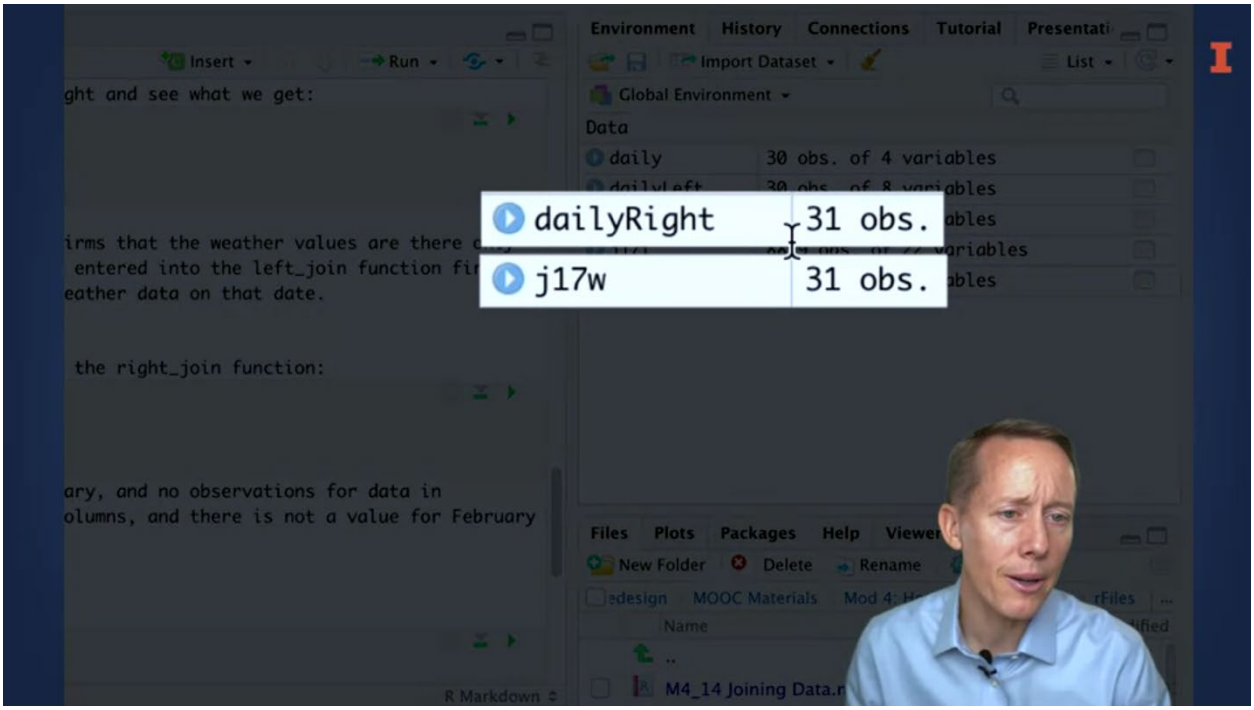


	date	avgCost	maxPrice	transactionQuantity	PRCP	SNOW	TMAX	TMIN
16	2017-01-18	-0.8408000	23.35	106	3.5039370	0.0000000	46.04	32.00
17	2017-01-19	0.8724194	24.39	93	6.6141732	0.0000000	44.96	35.06
18	2017-01-20	0.1718182	24.60	102	6.6929134	0.3149606	39.92	32.00
19	2017-01-21	0.1102558	24.30	161	0.0000000	0.0000000	44.96	24.98
20	2017-01-22	0.1095670	24.45	168	0.0000000	0.0000000	48.02	30.92
21	2017-01-23	0.1108696	18.43	8	5.9842520	1.8110236	44.96	32.00
22	2017-01-24	0.1102020	24.27	109	0.0000000	0.0000000	33.08	21.02
23	2017-01-25	0.2696089	24.55	107	0.0000000	0.0000000	33.98	15.98
24	2017-01-26	0.5272794	24.10	219	0.5118110	0.7874016	42.98	26.96
25	2017-01-27	0.6962763	24.27	247	0.0000000	0.0000000	42.98	24.98
26	2017-01-28	0.5549882	24.04	149	0.0000000	0.0000000	42.08	24.08
27	2017-01-29	0.1230079	24.60	148	0.0000000	0.0000000	44.96	24.98
28	2017-01-30	0.1100000	21.29	14	0.0000000	0.0000000	44.96	24.98
29	2017-01-31	0.1103239	24.04	101	0.0000000	0.0000000	42.98	26.06
	2017-02-01	0.1103478	24.30	77	NA	NA	NA	NA

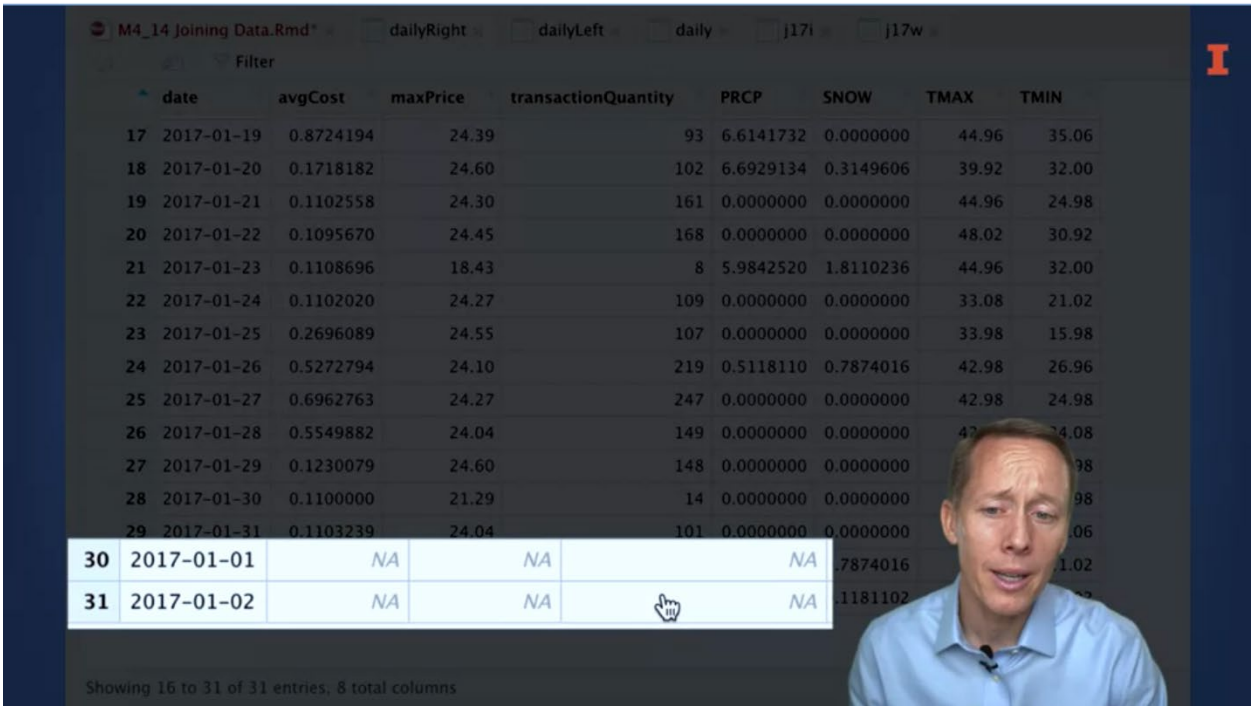
So why does the dailyleft data frame only have 30 observations? Well, a left join means we're only keeping the observations in the left data frame in this case, the daily data. And if there is not a matching value in the weather data, the right data frame, then it will just fill in those values with any, all right? And also, even though we had a January 1st in January 2nd observation in the weather data. We're not adding in those values here, all right? So left join only keeps the observations that are in the left data frame.



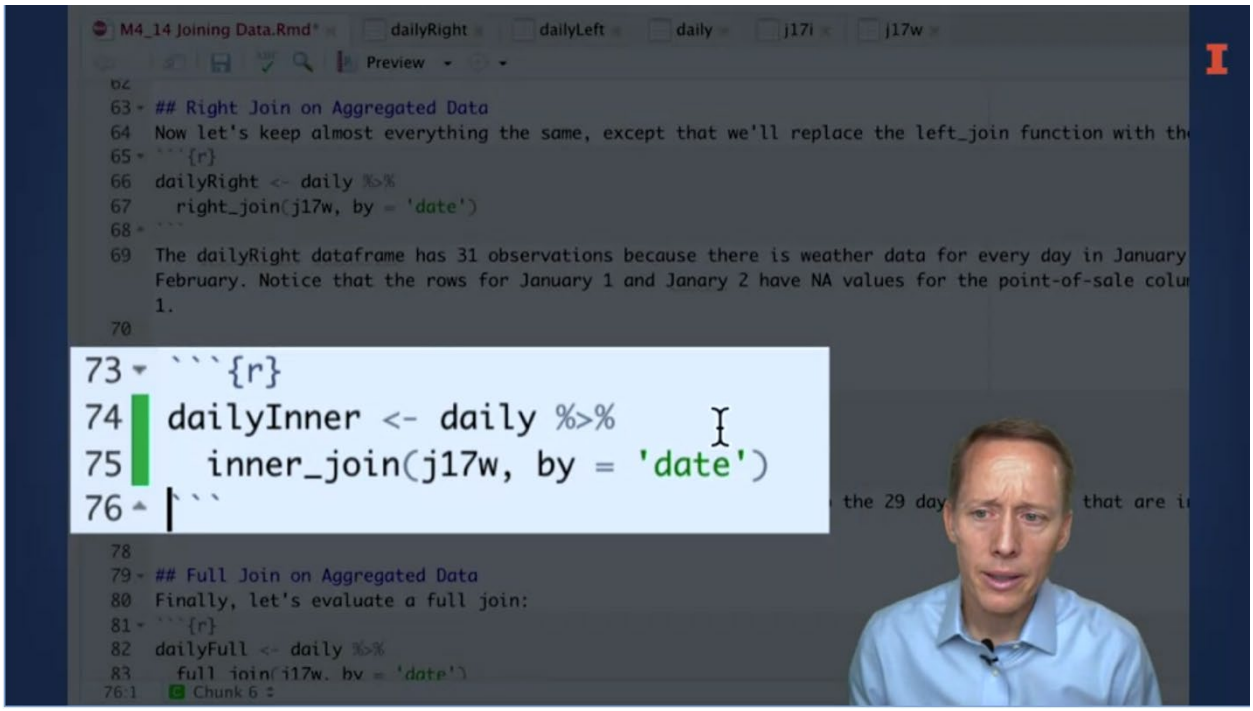
Now In contrast to the left join, there is a right join, and it's very parallel to the left join. So I'll create I'll illustrate the right, join by creating this dailyRight data frame object. And it's basically identical to the left join, except that I'm using the right join function here. Now let's go ahead and do that, and then just look at the dimensions in this case.



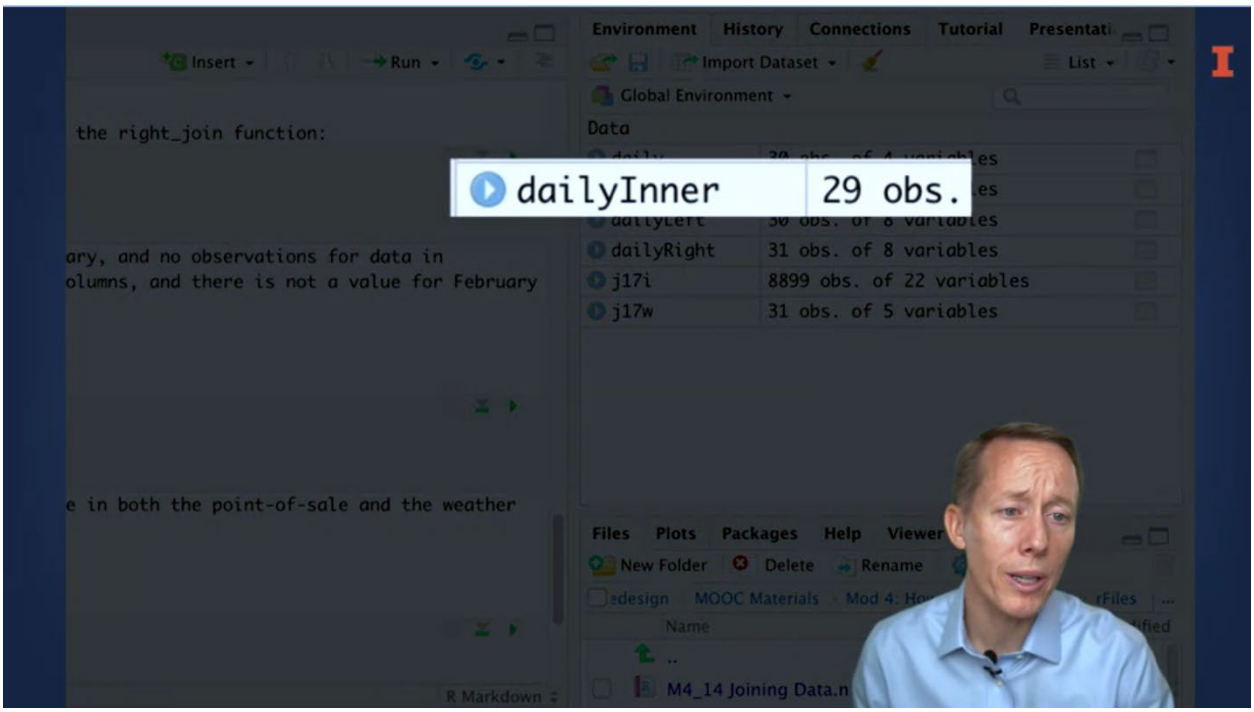
Now I've got 31 observations, also of eight variables.



And it looks very similar, except that now I'm only keeping observations in the left data frame if there's a matching value in the right data frame. In this case the right data frame is the weather data, so you can see it starts on the third and goes down to the 31st. And notice it didn't keep the February 1st observation, but it does keep the January 1st in January 2nd observations from the weather data. But there's no matching column information for the daily data frame, all right?



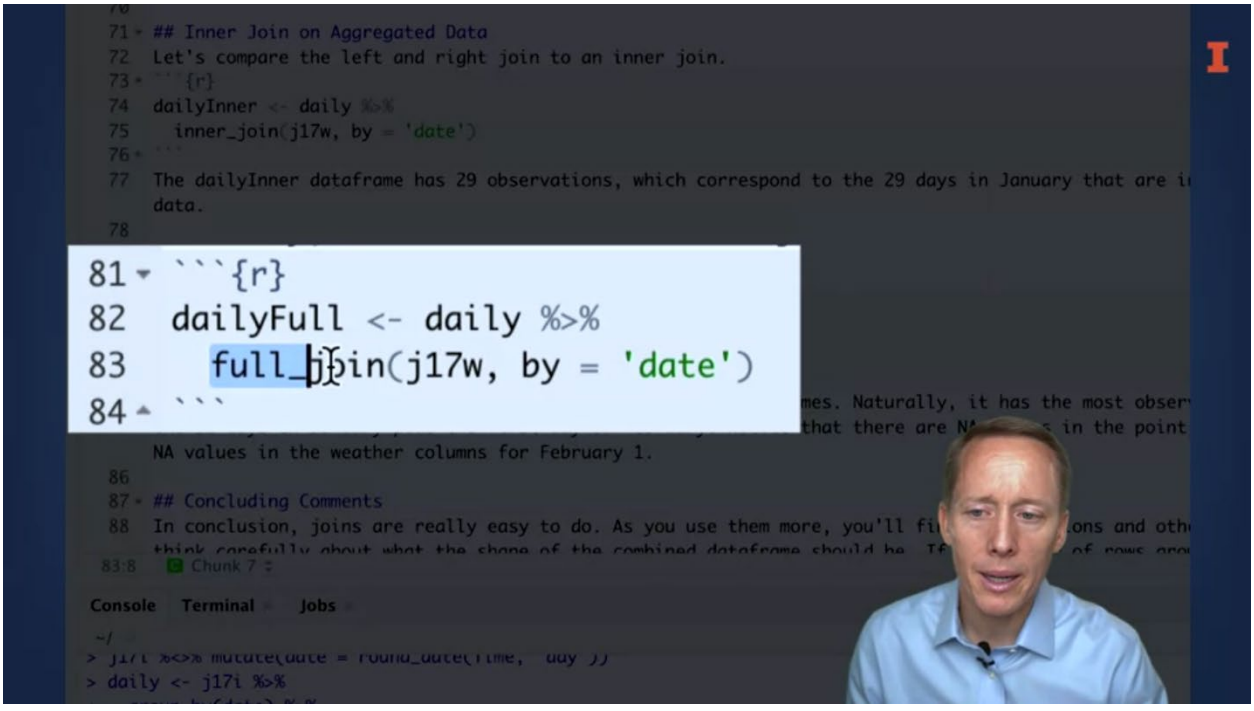
So that's a right joint, all right? Now let's compare that to an inner join. And again, it looks very similar to the left join and the right join but now I'm just using the inner join function.



And when I run that, and look at the number of observations I've only got 29 observations of eight variables.

	date	avgCost	maxPrice	transactionQuantity	PRCP	SNOW	TMAX	TMIN
1	2017-01-03	0.1135294	24.10	4	0.0000000	0.0000000	23.00	6.98
2	2017-01-04	0.1896842	23.59	95	0.0000000	0.0000000	21.92	3.92
3	2017-01-05	0.1102067	24.60	133	0.0000000	0.0000000	28.94	8.96
4	2017-01-06	0.1568137	24.63	132	0.0000000	0.0000000	35.06	12.92
5	2017-01-07	0.1104154	24.45	133	0.0000000	0.0000000	39.02	17.06
6	2017-01-08	0.1103021	23.47	117	6.4960630	0.7086614	39.92	32.00
7	2017-01-09	0.1100000	14.68	8	0.5905512	0.3149606	44.06	30.92
8	2017-01-10	-0.7733178	24.45	83	0.0000000	0.0000000	46.94	26.96
9	2017-01-11	0.1101481	24.27	102	0.0000000	0.0000000	53.06	30.02
10	2017-01-12	0.1591909	24.30	119	0.0000000	0.0000000	48.0	30.02
11	2017-01-13	0.7336964	23.59	112	0.0000000	0.0000000		08
12	2017-01-14	0.5803980	24.39	154	0.0000000	0.0000000		98
13	2017-01-15	0.1102649	23.59	147	0.0000000	0.0000000		94
14	2017-01-16	0.1103636	18.43	18	0.0000000	0.0000000		9.04
15	2017-01-17	-0.7083983	21.70	82	0.1181102	0.0000000		00
16	2017-01-18	-0.8408000	23.35	106	3.5039370	0.000		
17	2017-01-19	0.8724194	24.39	93	6.6141732	0.0		

So an inner join, if we look at the values in here, it only retains observations from both data frames. If there's a matching value in both data frames, so in this case it only goes from January 3rd to January 31st. It doesn't include, the February 1st observation from the daily data frame. Or the January 1st and January 2nd observations from the weather data frame, okay, so that's an inner join.



Now the last join I want to illustrate is a full_join, and at this point you can probably

guess what a full_join is going to do. And it looks very similar to the other joins, except that I'm using the full_join function.

	date	avgCost	maxPrice	transactionQuantity	PRCP	SNOW	TMAX	TMIN
18	2017-01-18	-0.8408000	23.35	106	3.5039370	0.0000000	46.04	32.00
19	2017-01-19	0.8724194	24.39	93	6.6141732	0.0000000	44.96	35.06
20	2017-01-20	0.1718182	24.60	102	6.6929134	0.3149606	39.92	32.00
21	2017-01-21	0.1102558	24.30	161	0.0000000	0.0000000	44.96	24.98
22	2017-01-22	0.1095670	24.45	168	0.0000000	0.0000000	48.02	30.92
23	2017-01-23	0.1108696	18.43	8	5.9842520	1.8110236	44.96	32.00
24	2017-01-24	0.1102020	24.27	109	0.0000000	0.0000000	33.08	21.02
25	2017-01-25	0.2696089	24.55	107	0.0000000	0.0000000	33.98	15.98
26	2017-01-26	0.5272794	24.10	219	0.5118110	0.7874016	42.98	26.96
27	2017-01-27	0.6962763	24.27	247	0.0000000	0.0000000	42.98	24.98
28	2017-01-28	0.5549882	24.04	149	0.0000000	0.0000000	42.08	24.08
29	2017-01-29	0.1230079	24.60	148	0.0000000	0.0000000	44.96	24.98
30	2017-01-30	0.1100000	21.29	14	0.0000000	0.0000000	44.96	24.98
31	2017-01-31	0.1103239	24.04	101	0.0000000	0.0000000	42.98	26.06
32	2017-02-01	0.1103478	24.30	77	NA	NA	NA	NA

Showing 17 to 32 of 32 entries, 8 total columns

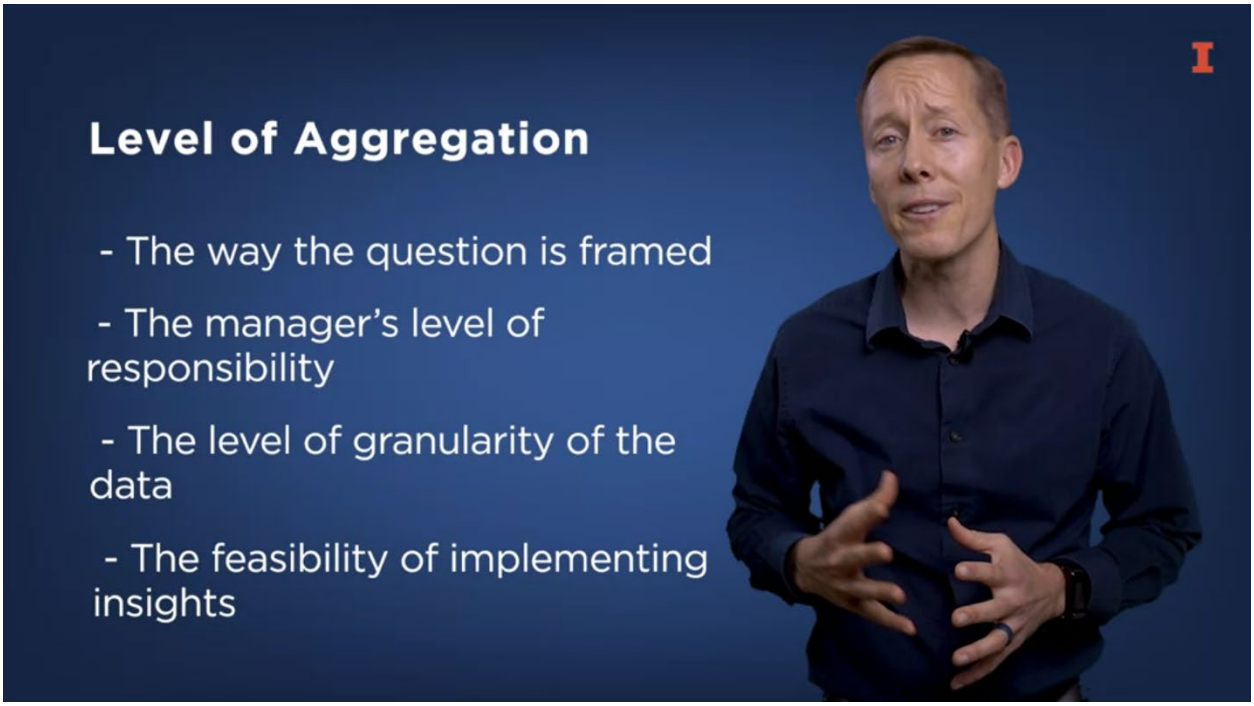
So I'll run that, and let's just look at the dimensions for a full_join, it's got 32 observations. If I open that up and I'm going to sort on date here. We start with January 1st and January 2nd, from the weather data frame, and we go all the way down to February 1st from the daily data frame. And there's missing values in the weather there. So a full join retains all observations in both data frames. So there are for examples of combining two data frames using different join functions. I would just recommend that you think carefully about what join you want to use. And make sure that the resulting number of rows is what you expect it to be.

Lesson 4-1.15 Module 4 Conclusion

Now that you've completed this module, you should have a deeper understanding of the procedure required to assemble data. While tidying up the data is an important part of the data assembly procedure, there is additional pre-processing that typically occurs before data is processed with analytic tools to find actionable insight.

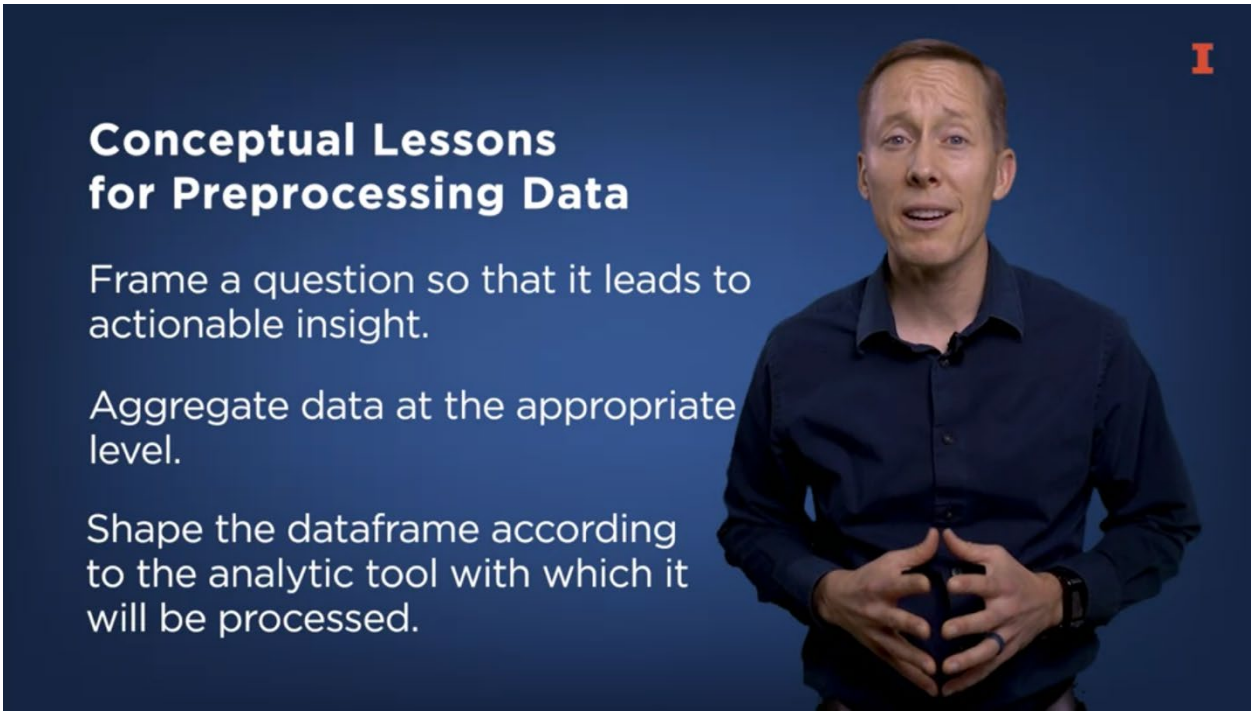
Remember that just like clay has to be pre-processed before it can be processed by a kiln, into a useful vase. Data frames have to be pre-processed before they can be processed into actionable insights.

The conceptual lessons in this module focused on issues that should influence the extent to which data is aggregated, and the shape of the data frame. Do not forget to frame a question so that the data analyses lead to actionable insight. The way a question is framed has a big influence on the level at which the data is aggregated.

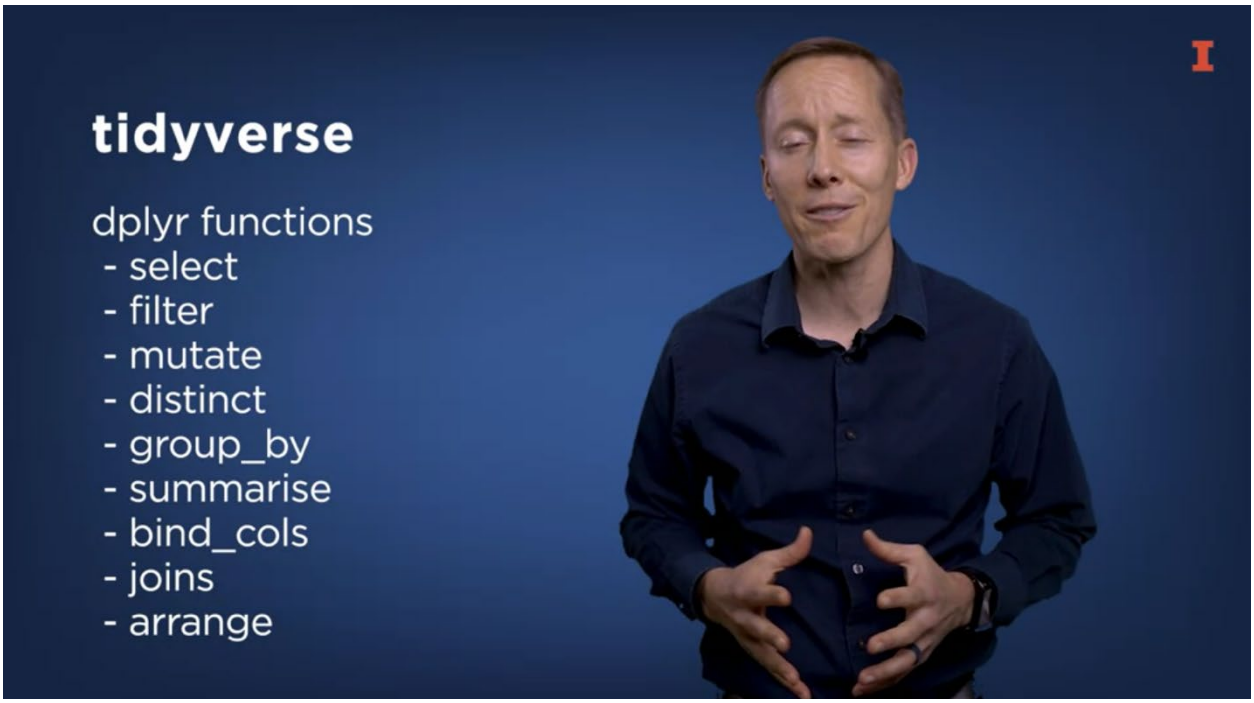


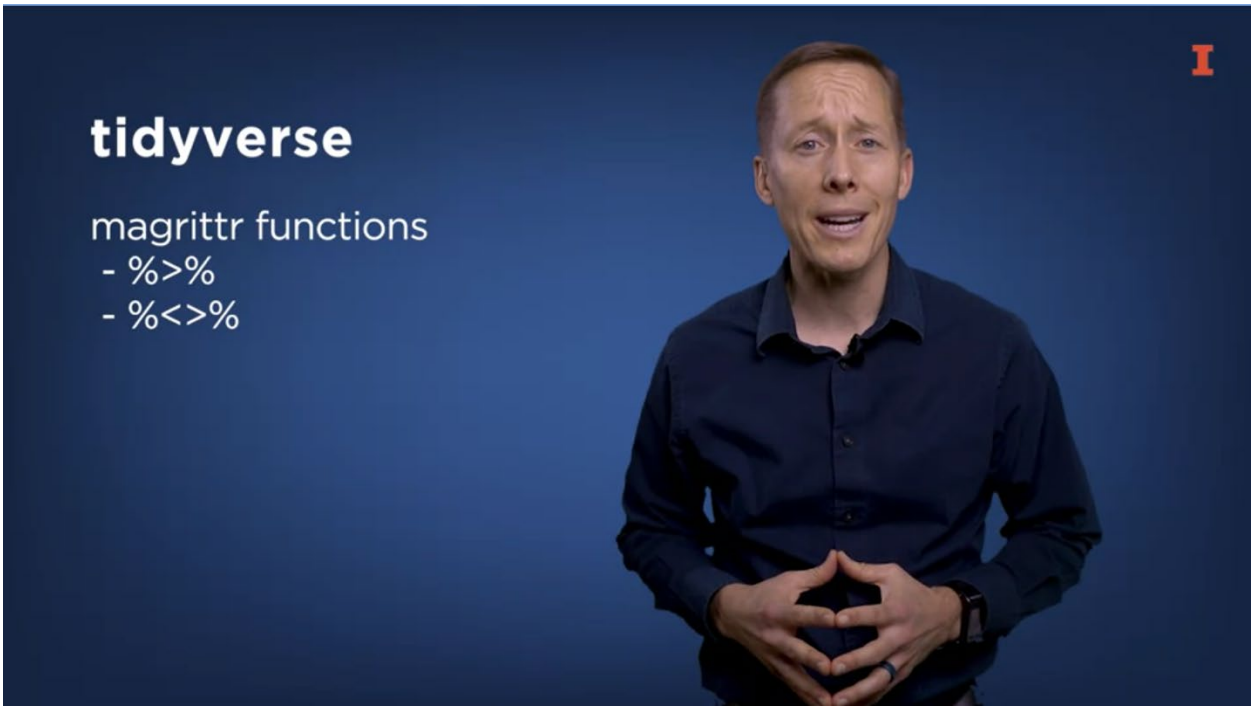
Another factor that should influence the level at which the data is aggregated is the managers level of responsibility. For instance, a product level manager is probably not

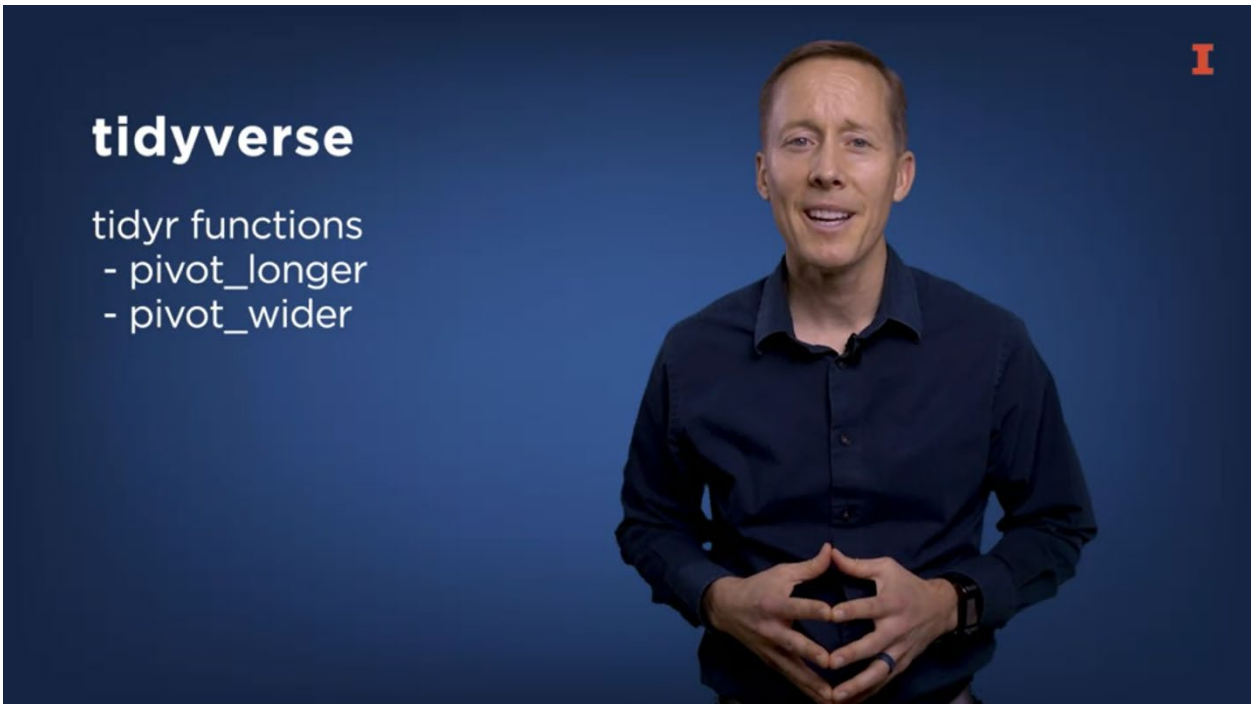
going to get much use out of data that is aggregated at the level of sales agents. The level of granularity of the raw data also influences the extent to which data can be aggregated, because you can't disaggregate data below the level at which the raw data is aggregated. For instance, you can't analyze daily sales trends if the raw data is aggregated at the monthly level. It's also important to recognize that even if the raw data is very granular, such that you can gain insight into minute details, you have to balance the desire to control small details with the feasibility of doing so. Shaping the data frame appropriately is another aspect of pre-processing.



The shape of the data frame will be heavily influenced by the tool that you will use to analyze the data. Wide data has many columns, and long data has many rows. Wide data is often most useful for algorithms so that you can evaluate business results within a detailed context. Long data is often most useful for dashboards and visualizations.


tidyverse
dplyr functions
- select
- filter
- mutate
- distinct
- group_by
- summarise
- bind_cols
- joins
- arrange


tidyverse
magrittr functions
- %>%
- %<>%



The practical lessons in this module focused on using functions from several packages in the tidyverse set of packages. One of the most useful tidyverse packages is dplyr. This package contains functions such as select, filter, mutate, distinct group_ by, summarize, bind_cols, joins and arrange. We also demonstrated some functions from other packages in the tidyverse. Specifically, the pipe operator from the McGregor package is simple but powerful because it allows you to change multiple pre-processing functions together in an easily readable way. We also demonstrated the pivot_longer, and pivot_wider functions from the tidyr package. These functions help us reshape data into the format that's appropriate for the data analytic tools that we will use to process the data. Now that you're equipped with this conceptual and practical knowledge, I hope that you're able to identify the necessary steps for preparing data sets to insert into data analytic tools.


References
https://pixabay.com/photos/pottery-kitchen-utensils-1956198/