

## Module 4: How Do I Preprocess Data?

### Table of Contents

<b>Module 4: How Do I Preprocess Data? .....</b>	<b>1</b>
<b>Lesson 4-0: Module 4 Overview.....</b>	<b>2</b>
Lesson 4-0.1 Introduction .....	2
<b>Lesson 4-1: Lectures .....</b>	<b>10</b>
Lesson 4-1.1 Introduction .....	10
Lesson 4-1.2 Framing Questions for Actionable Insight.....	11
Lesson 4-1.3 Dataframe Shape: Level of Aggregation.....	22
Lesson 4-1.4 Dataframe: Control Versus Feasibility .....	31
Lesson 4-1.5 Dataframe Shape: Wide Versus Long.....	39
Lesson 4-1.6 Review of Notebooks and Introduction to dplyr.....	46
Lesson 4-1.7 Subset Data Using dplyr's Select and Filter Functions .....	55
Lesson 4-1.8 Useful Operators: %.% and %in% .....	71
Lesson 4-1.9 Using dplyr's Mutate, Rename, Relocate, and Distinct Functions .....	85
Lesson 4-1.10 Handling Missing Values .....	100
Lesson 4-1.11 Data Aggregation and Summary.....	112
Lesson 4-1.12 Pivoting Dataframes Between Wide and Long Shapes.....	118
Lesson 4-1.13 Stacking and Sorting Data .....	133
Lesson 4-1.14 Joining Data .....	148
Lesson 4-1.15 Module 4 Conclusion.....	162

## Lesson 4-0: Module 4 Overview

### Lesson 4-0.1 Introduction



2017bluebudgie/ Public Domain / Pixabay / vase-2848243

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

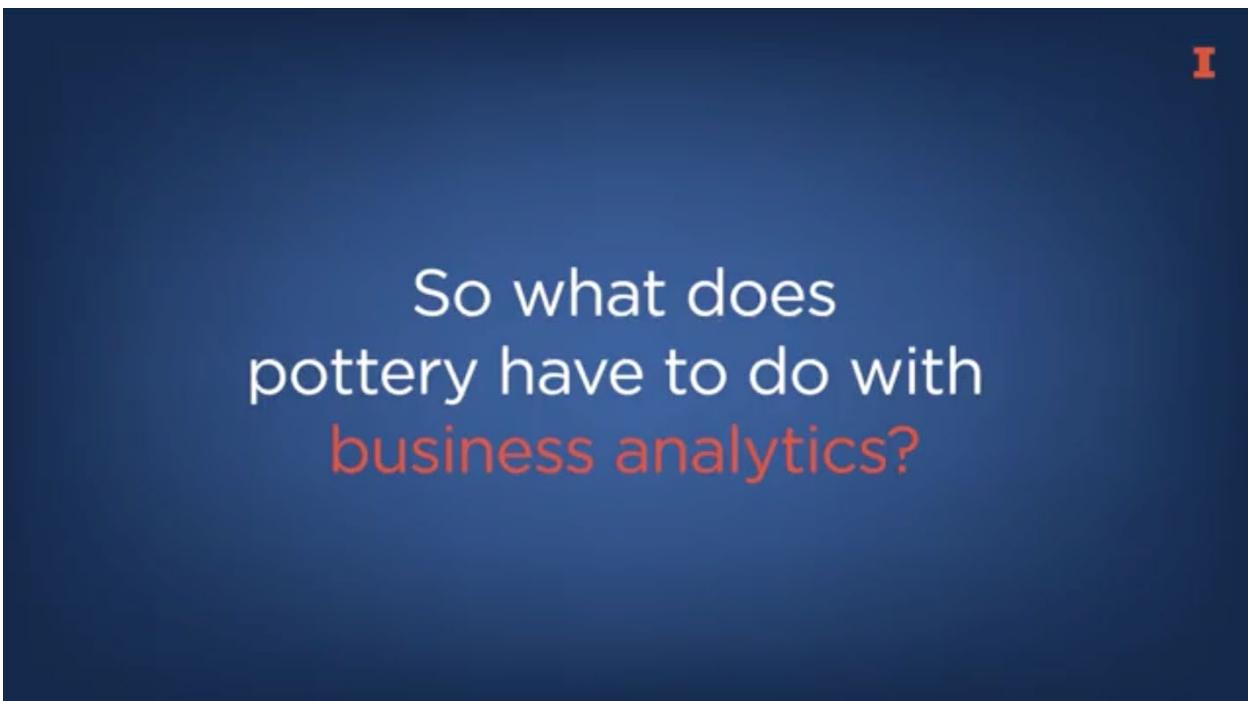
before it is glazed and then put into a kiln and fired. The end goal is to turn the slab of clay into a product that is both beautiful and useful.



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 have to do with business analytics?

## FACT Framework

- Frame the question
- Assemble the data
- Calculate the results
- Tell others the results



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

## Data Assembly Steps

Tidy up the dataframe

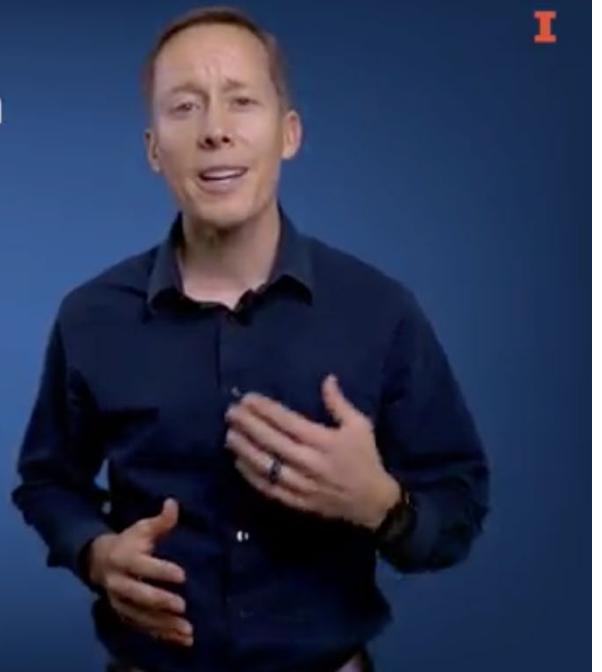


Tidying up the data is similar to removing the impurities from the clay.

## Preprocess the Data

Aggregate the data at the appropriate level

Shape the dataframe to the appropriate length and width



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.

## Factors that Influence Data Processing

- Business question
- Managerial responsibility
- Level of granularity of the raw data
- Operational control verses operational feasibility
- Data analytic tool



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.

## Packages Introduced in this Module

- dplyr
- magrittr
- tidyr



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

## By the End of this Module

You will know how to reshape raw data frame into a variety of forms for processing.



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]. CCO 1.0. Pexels.

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

Shkraba, A.(2020). Video of personworking on pottery business[Video]. CCO 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 video frame shows a man in a dark blue button-down shirt standing against a dark blue background. He is gesturing with his hands, palms facing up, as if explaining something. In the top right corner of the slide, there is a small red letter 'I'.

## FACT Framework

- Frame the question
- Assemble the data
- Calculate the results
- Tell others the results

RONALD GUYMON  
Senior Lecturer of Accountancy

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.



LUIS GUILAMO

Dir. of Analytics and Application Development

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 waivers.

## Three Lessons About Framing Questions

1. Break down a big problem into smaller pieces.



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.

## Breaking Down the NFL Draft into Smaller Pieces

1. Understanding the field, meaning the available players
2. Understanding your own roster, or the players who are currently on your team, and
3. Understanding everybody else's roster, or the players who are on other teams

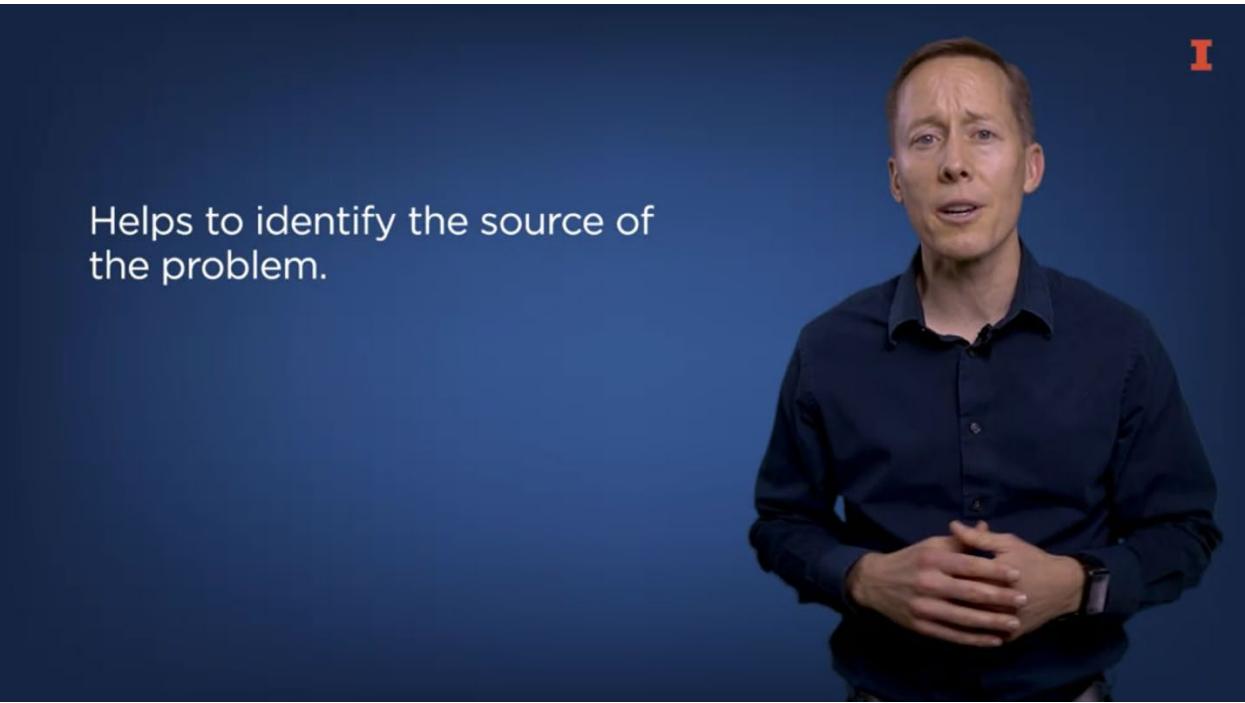


Then he described the three pieces; one, understanding the field, meaning the available players, two understanding your own roster or the players who are currently on your team, and three, understanding everybody else's roster or the players who are on other teams.



Speeds up the analysis by allowing for multiple people to work on the task.

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.



Helps to identify the source of the problem.

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.

## Three Lessons About Framing Questions

1. Break down a big problem into smaller pieces
2. Consider the level of aggregation for the analysis
3. Make sure that the question is focused on a business goal.



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.

## Luis's NFL Draft Question

**Version 1:** How can we identify the best players?

Further refine the question so that it encourages a careful and methodical approach that will lead to actionable insight.



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.



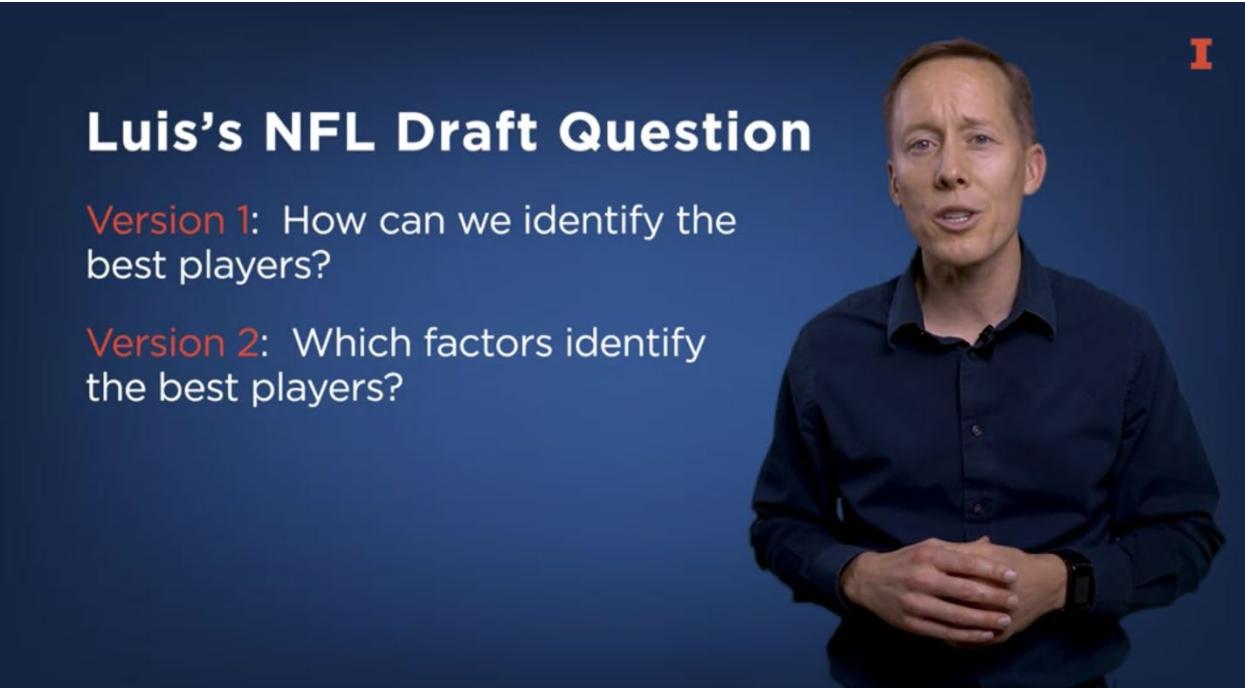
This will take curiosity and a deeper understanding of data and how to analyze it.

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

## Luis's NFL Draft Question

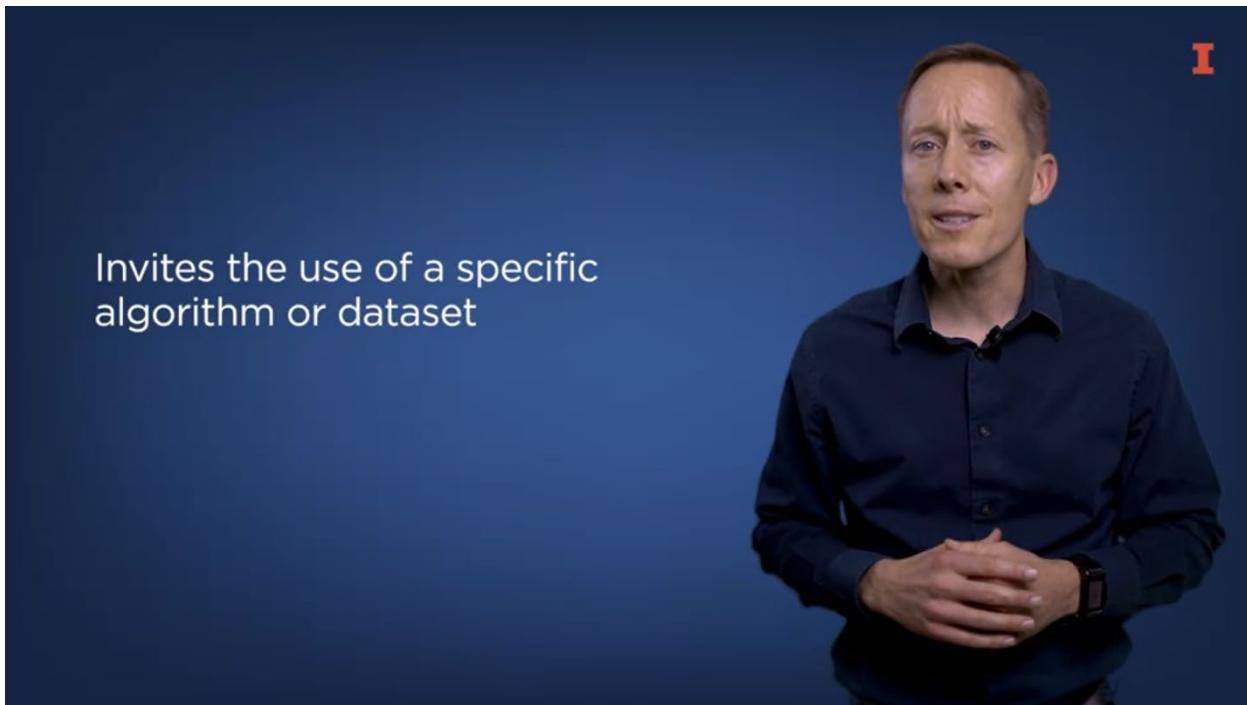
**Version 1:** How can we identify the best players?

**Version 2:** Which factors identify the best players?



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.



Invites the use of a specific algorithm or dataset

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?

## Luis's NFL Draft Question

**Version 1:** How can we identify the best players?

**Version 2:** Which factors identify the best players?

**Version 3:** 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.

Outcome of our analysis should lead to actionable insight, so **domain knowledge** is a critical ingredient of data analysis.



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

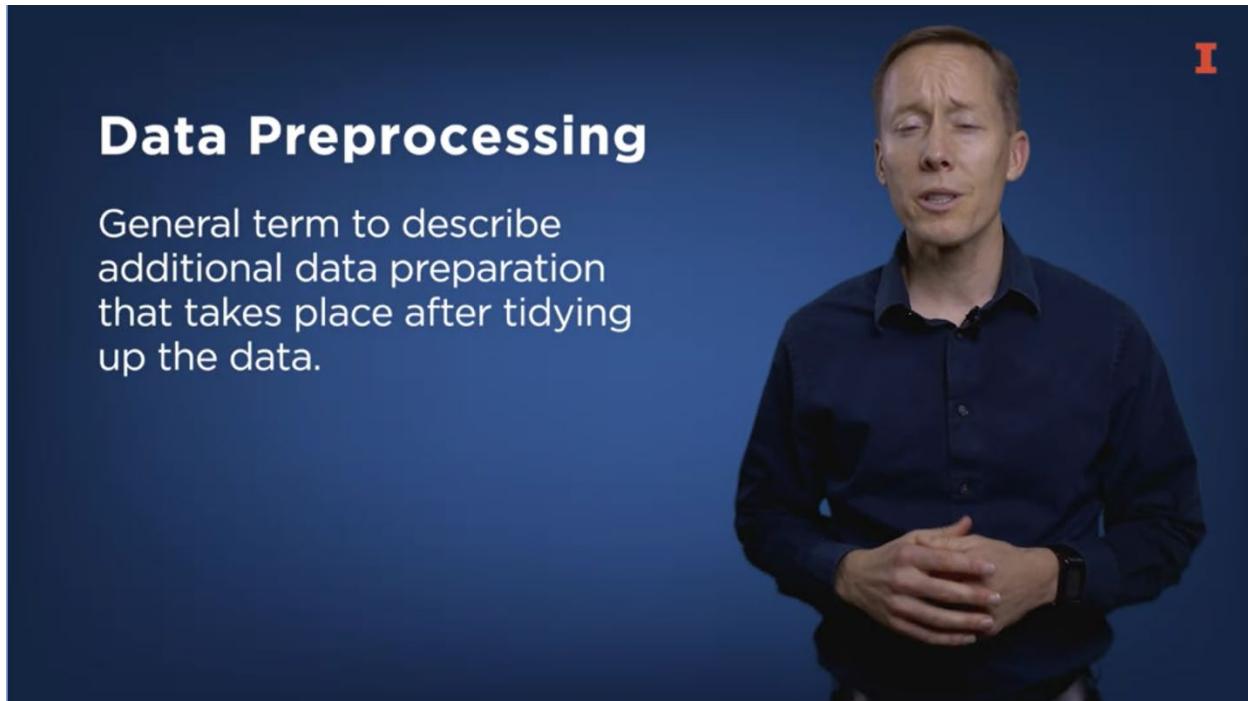
that we want to simply create a rank ordering of the players and pick them based on the highest ranked player available for each turn.

**Luis's NFL Draft Question**

**Version 4:** What factors lead to a rank ordering of players based on expected career earnings?

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



## Data Preprocessing

General term to describe additional data preparation that takes place after tidying up the data.

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.



2017 StockSnap / Public Domain / Pixabay / people-2588030

2015 Courtany / Public Domain / Pixabay / boy-773630

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.

## Three Factors That Influence Data Aggregation

1. The manager's decision domain



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.

## Three Factors That Influence Data Aggregation

1. The manager's decision domain
2. The way in which the question is framed



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.

## Three Factors That Influence Data Aggregation

1. The manager's decision domain
2. The way in which the question is framed
3. The level of granularity of the raw data



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.

The image displays three vertically stacked tables, each with a header row and several data rows. The first table is titled 'Category' and lists items like Aubergine and Chickpea Vindaloo, Beef, and Beef Stew. The second table is titled 'Department' and lists categories like Beverage, Catering, Entrees, Salad, and Sides. The third table is titled 'CardholderName' and lists individuals like Aalyah McQueeney, Adamaris Vergari, Kaden Tiell, Shacole Seneca, and Tayce Fortes. Each table has columns for Cost, Revenue, and Discounts.

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.

## Three Factors That Influence Data Aggregation

1. The manager's decision domain
2. The way in which the question is framed
3. The level of granularity of the raw data



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.

## References

- StockSnap. (2017). people-2588030 [Online image]. CCO 1.0. Pixabay.
- Courtany. (2015). boy-773630 [Online image]. CCO 1.0. Pixabay.
- PublicDomainPictures. (2012). black-72856 [Online image]. CCO 1.0. Pixabay.
- mohamed\_hassan. (2018). teamwork-3499960 [Online image]. CCO 1.0. Pixabay.
- No-longer-here. (2013). hairdryer-163576 [Online image]. CCO 1.0. Pixabay.

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.

## Actionable Insight

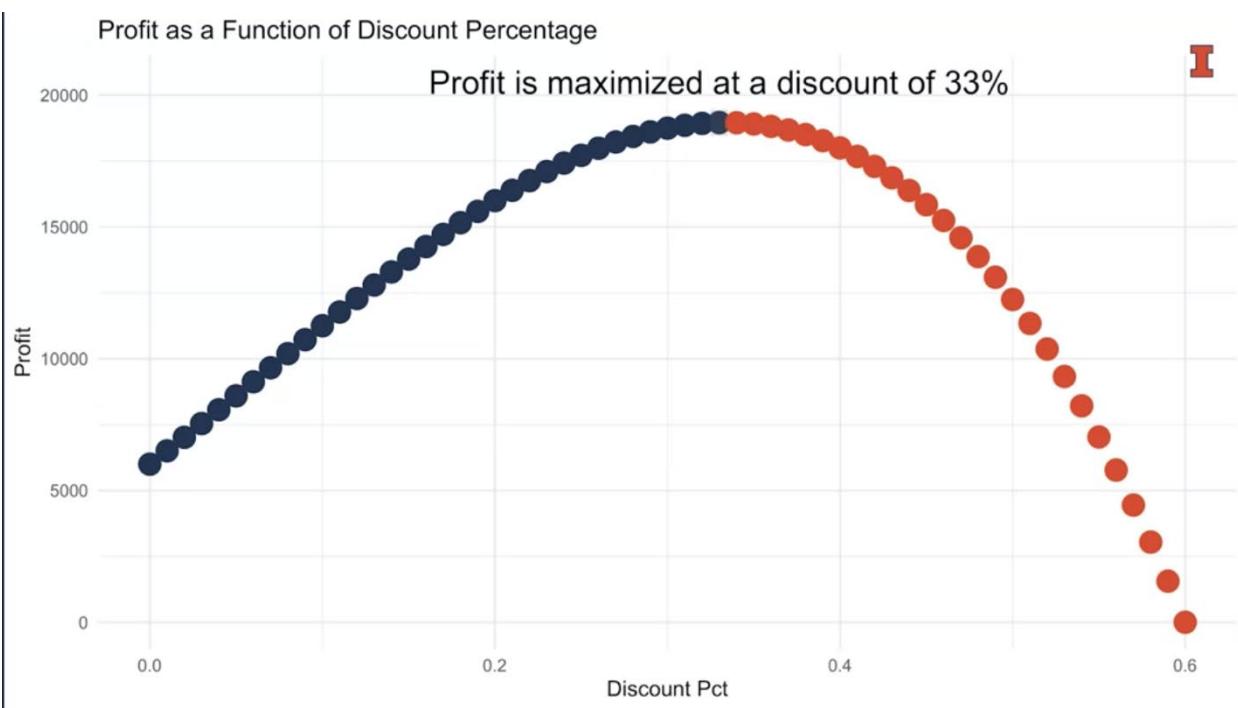
An insight that (1) will further the organization's goals, and (2) is operationally feasible

Actionable insights are specific.

Specific insights depend on disaggregated data.



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.

## Price Elasticity of Demand

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.

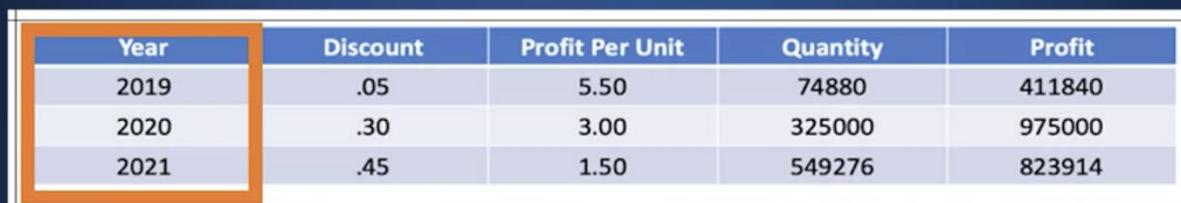
Hour	Discount	Profit Per Unit	Quantity	Profit
2021-11-07 13:00	.05	5.50	14	77.00
2021-11-07 14:00	.30	3.00	62	186.00
2021-11-07 15:00	.45	1.50	105	157.50

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.



When data is too granular,  
it can lead to insight that  
requires too much effort to  
be actionable.

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.

When data is too aggregated, it leads to insight, but it does not lead to many actions.

Another problem with data that is too aggregated is that the details are lost.



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.

## In Summary

When seeking for actionable insight, there's a tradeoff between control and feasibility.

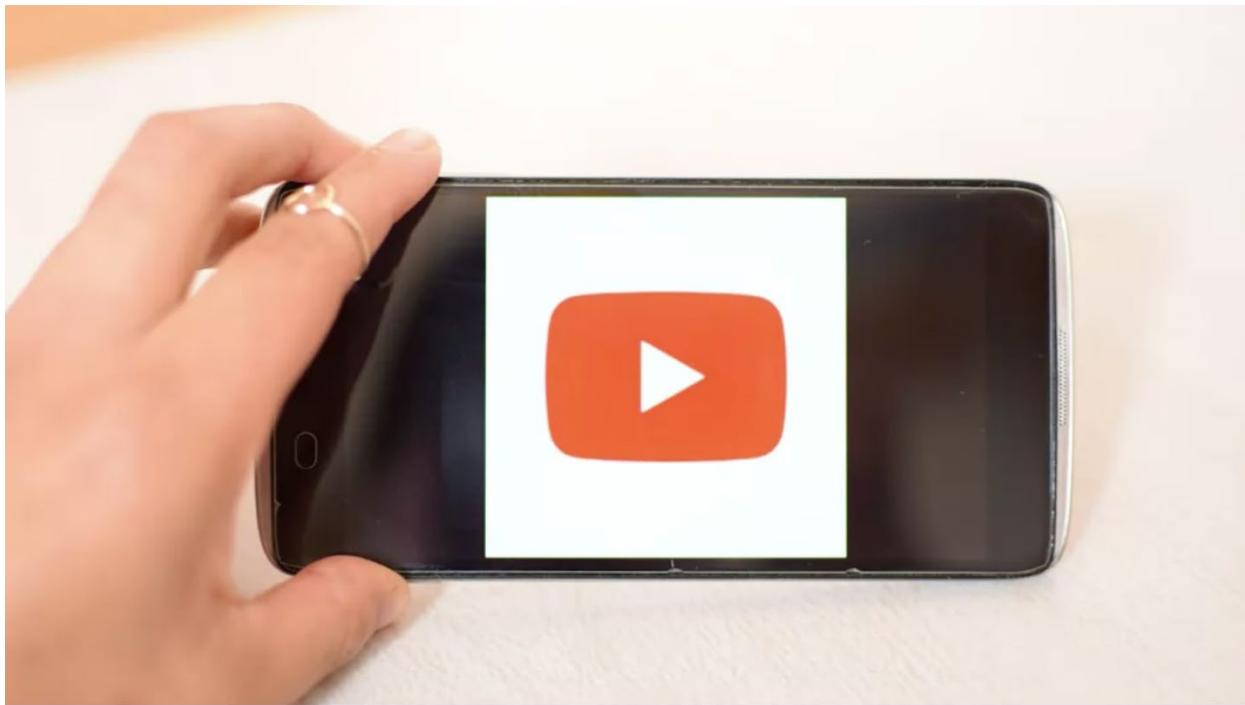
When the data is too granular, the insight offers a lot of control but is infeasible.

When the data is too aggregated, the insight is feasible but does not increase control.



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.

## Business Analytics

The length and width of a dataframe depends on the tool that you use to analyze the data.

What does the length and width of the data mean?



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?

## Length & Width

The length of a dataframe refers to the number of observations.

The width of a dataframe refers to the number of columns.

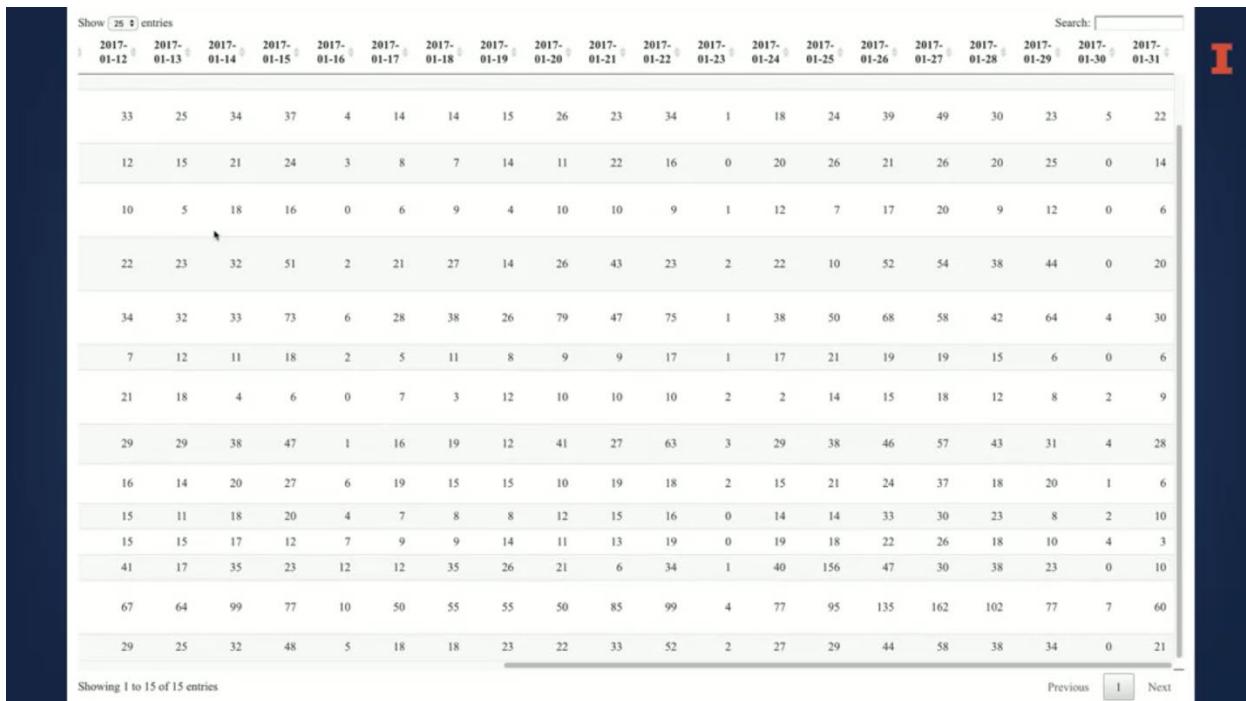
A **long** dataframe has many **rows**

A **wide** dataframe has many **columns**



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.



The screenshot shows a data grid with 15 rows and 31 columns. The columns are labeled with dates from January 12 to January 31, 2017. The rows are labeled with numbers 1 through 15, representing different line items. Each cell in the grid contains a numerical value representing sales quantity. The interface includes a search bar at the top right and navigation buttons at the bottom right.

	2017-01-12	2017-01-13	2017-01-14	2017-01-15	2017-01-16	2017-01-17	2017-01-18	2017-01-19	2017-01-20	2017-01-21	2017-01-22	2017-01-23	2017-01-24	2017-01-25	2017-01-26	2017-01-27	2017-01-28	2017-01-29	2017-01-30	2017-01-31
1	33	25	34	37	4	14	14	15	26	23	34	1	18	24	39	49	30	23	5	22
2	12	15	21	24	3	8	7	14	11	22	16	0	20	26	21	26	20	25	0	14
3	10	5	18	16	0	6	9	4	10	10	9	1	12	7	17	20	9	12	0	6
4	22	23	32	51	2	21	27	14	26	43	23	2	22	10	52	54	38	44	0	20
5	34	32	33	73	6	28	38	26	79	47	75	1	38	50	68	58	42	64	4	30
6	7	12	11	18	2	5	11	8	9	9	17	1	17	21	19	19	15	6	0	6
7	21	18	4	6	0	7	3	12	10	10	10	2	2	14	15	18	12	8	2	9
8	29	29	38	47	1	16	19	12	41	27	63	3	29	38	46	57	43	31	4	28
9	16	14	20	27	6	19	15	15	10	19	18	2	15	21	24	37	18	20	1	6
10	15	11	18	20	4	7	8	8	12	15	16	0	14	14	33	30	23	8	2	10
11	15	15	17	12	7	9	9	14	11	13	19	0	19	18	22	26	18	10	4	3
12	41	17	35	23	12	12	35	26	21	6	34	1	40	156	47	30	38	23	0	10
13	67	64	99	77	10	50	55	55	50	85	99	4	77	95	135	162	102	77	7	60
14	29	25	32	48	5	18	18	23	22	33	52	2	27	29	44	58	38	34	0	21

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.

## Width & Length

Visualizations and dashboards often rely on long dataframes.

Algorithms and human-readable tables often rely on wide dataframes.

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.

## Summary

Assembling data requires:

- Tidying up the dataframe
- Aggregation at the appropriate level
- Shaping the data to the appropriate length and width



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]. CCO 1.0. Pixabay.
- nikuga. (2020). tiktok-5390055 [Online image]. CCO 1.0. Pixabay.
- zsuzsannasolti. (2017). pottery-1956198 [Online image]. CCO 1.0. Pixabay.

## Lesson 4-1.6 Review of Notebooks and Introduction to dplyr

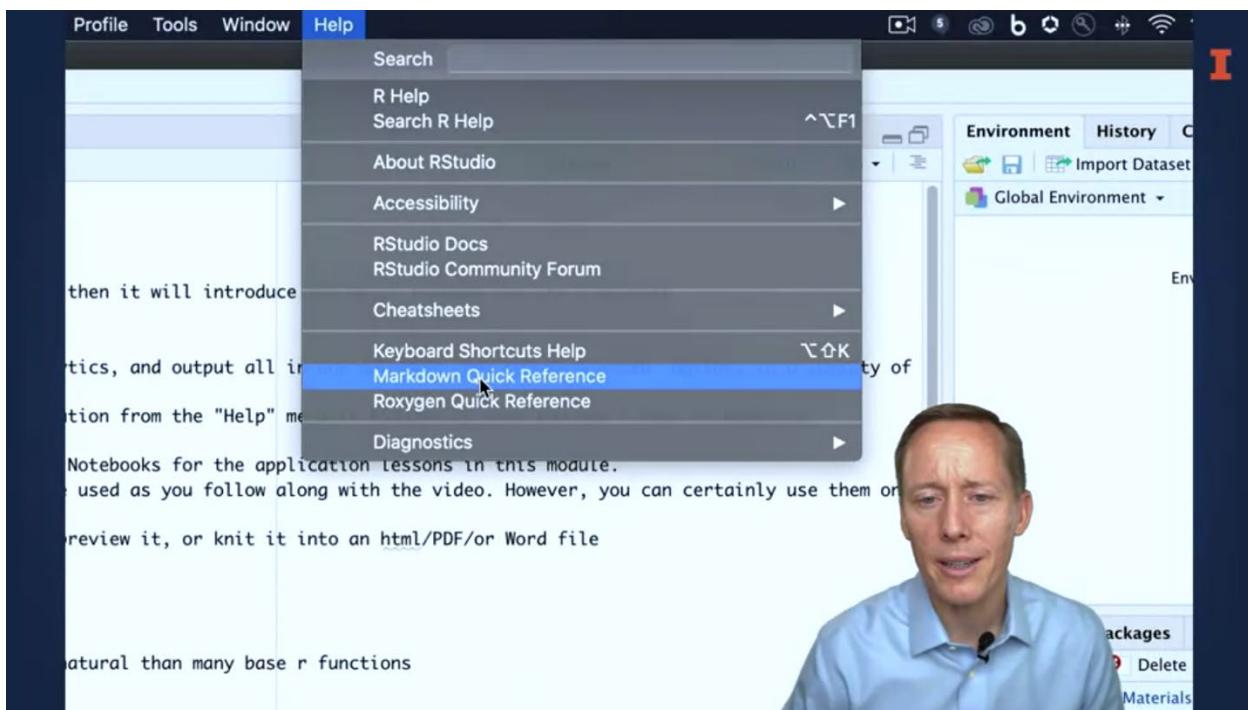
```
16 2. Reduces the amount of verbiage required to write code
19 3. Harmonizes with other packages in the Tidyverse
20
21 If you haven't already installed the dplyr package on the machine that you're using, then you can
If you've already installed the package then you may get an error.
22 ````{r}
23 install.packages('dplyr')
24 ````

25
26 You only need to install the package once, but every time you begin a new R session, you need to l
command:
27 ````{r}
28 library(dplyr)
29 ````

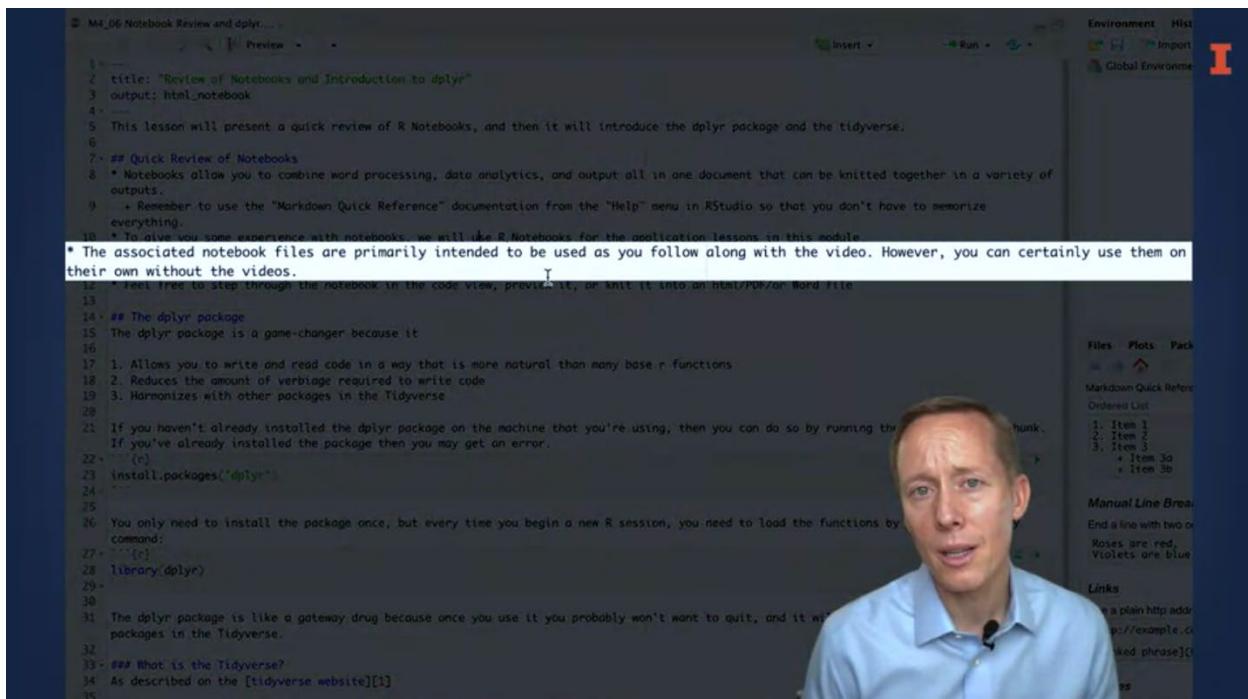
30
31 The dplyr package is like a gateway drug because once you use it you probably won't want to quit,
packages in the Tidyverse.
32
33 ## What is the Tidyverse?
34 As described on the [tidyverse website][1]
35
36 > The tidyverse is a collection of R packages designed for data science. These packages share an und
data structures.
37
38 Some packages that you may already be familiar with include:
39
28:15 0 Chunk 2 ▾
```

Console

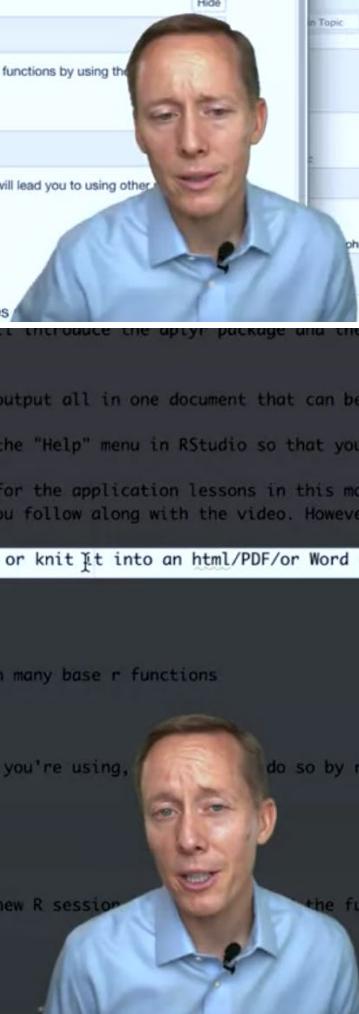
In this lesson, I want to quickly review our notebooks and also introduce you to the dplyr package as well as the tidyverse. I want to review our notebooks with you because up to this point the lessons have used our script files. Now in this lesson and all the other lessons in this module, we're going 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.



The dplyr package Review and dplyr.html Open in Browser Find Publish Options Tutorial Press

M4\_06 Notebook Review and dplyr.html

- Notebooks allow you to combine word processing, data analytics, and output all in one document that can be knitted together in a variety of outputs.
  - Remember to use the "Markdown Quick Reference" documentation from the "Help" menu in RStudio so that you don't have to memorize everything.
- To give you some experience with notebooks, we will use R Notebooks for the application lessons in this module.
- The associated notebook files are primarily intended to be used as you follow along with the video. However, you can certainly use them on their own without the videos.
- Feel free to step through the notebook in the code view, preview it, or knit it into an html/PDF or Word file.

## The dplyr package

The dplyr package is a game-changer because it

- Allows you to write and read code in a way that is more natural than many base r functions
- Reduces the amount of verbiage required to write code
- Harmonizes with other packages in the Tidyverse

If you haven't already installed the dplyr package on the machine that you're using, then you can do so by running the following code chunk. If you've already installed the package then you may get an error.

```
install.packages('dplyr')
```

You only need to install the package once, but every time you begin a new R session, you need to load the functions by using the command:

```
library(dplyr)
```

The dplyr package is like a gateway drug because once you use it you probably won't want to quit, and it will lead you to using other packages in the Tidyverse.

## What is the Tidyverse?

As described on the tidyverse website

The tidyverse is a collection of R packages designed for data science. All packages

```
## Quick Review of Notebooks
* Notebooks allow you to combine word processing, data analytics, and output all in one document that can be knitted together in a variety of outputs.
    + Remember to use the "Markdown Quick Reference" documentation from the "Help" menu in RStudio so that you don't have to memorize everything.
* To give you some experience with notebooks, we will use R Notebooks for the application lessons in this module.
* The associated notebook files are primarily intended to be used as you follow along with the video. However, you can certainly use them on their own without the videos.
* Feel free to step through the notebook in the code view, preview it, or knit it into an html/PDF or Word file

## The dplyr package
The dplyr package is a game-changer because it

1. Allows you to write and read code in a way that is more natural than many base r functions
2. Reduces the amount of verbiage required to write code
3. Harmonizes with other packages in the Tidyverse

If you haven't already installed the dplyr package on the machine that you're using, then you can do so by running the following command.
{r}
install.packages('dplyr')

You only need to install the package once, but every time you begin a new R session, you need to load the functions by using the command:
{r}
library(dplyr)
```

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.

```
14 ## The dplyr package
15 The dplyr package is a game-changer because it
16
17 1. Allows you to write and read code in a way that is more natural than many base r functions
18 2. Reduces the amount of verbiage required to write code
19 3. Harmonizes with other packages in the Tidyverse
20
21 If you haven't already installed the dplyr package on the machine that you're using, then you can do so by running the command:
22 - ``{r}
23 install.packages('dplyr')
24 -
25
26 You only need to install the package once, but every time you begin a new R session, you need to run the library command:
27 - ``{r}
28 library(dplyr)
29 -
30
31 The dplyr package is like a gateway drug because once you use it you, won't want to quit using other packages in the Tidyverse.
32
33 - ### What is the Tidyverse?
34 As described on the [tidyverse website][1]
35
36 > The tidyverse is a collection of R packages designed for data analysis, visualization, and modeling. They are an
```



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.

```
20
21 If you haven't already installed the dplyr pa I
22 If you've already installed the package then
23 ``{r}
24 install.packages('dplyr')
25
26 You only need to install the packa T
27 command:
28 ``{r}
29 library(dplyr)
30
31 The dplyr package is like beco
```



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?

The screenshot shows the tidyverse.org website. At the top, there's a navigation bar with links for Packages, Blog, Learn, Help, and Contribute. Below the navigation, there's a large image of several hexagonal icons representing different R packages like ggplot2, dplyr, and purrr. To the right of these icons, there's a section titled "R packages for data science" with a brief description and a code snippet for installing the tidyverse. Below this, there's a section titled "Learn the tidyverse" with a link to "R for Data Science". A video overlay of Professor Ronald Guymon is visible, speaking about the tidyverse. The video frame is semi-transparent, allowing the website content to be seen through it.

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("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.

The screenshot shows the 'Contents' page of the tidyverse.org website. At the top, there is a navigation bar with links for 'Packages', 'Blog', 'Learn', 'Help', and 'Contribute'. To the right of the navigation bar is a large red 'I' logo. Below the navigation bar, the word 'Contents' is centered. To the right of 'Contents', there is a vertical list of categories: 'Installation and use', 'Core tidyverse', 'Import', 'Wrangle', 'Program', 'Model', and 'Get help'. To the right of this list is a video player window showing a man in a blue shirt speaking. On the left side of the page, there is some text and code snippets related to tidyverse packages.

Install.packages("tidyverse").  
and make it available in your current R  
tidyverse.org.

Installation and use  
Core tidyverse  
Import  
Wrangle  
Program  
Model  
Get help



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

The screenshot shows the 'Core tidyverse' page. At the top, the title 'Core tidyverse' is displayed with a link icon. To the right, there is a vertical list of categories: 'Installation and use', 'Core tidyverse', 'Import', 'Wrangle', 'Program', 'Model', and 'Get help'. Below the title, there is a brief description of the core tidyverse packages. To the right of the description is a video player window showing a man in a blue shirt speaking. On the left side, there are three package icons: ggplot2, dplyr, and tidyr, each with a yellow hexagon outline.

The core tidyverse includes the packages that you're likely to use in everyday data analyses. As of tidyverse 1.3.0, the following packages are included in the core tidyverse:

**ggplot2**  
ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details. Go to docs...

**dplyr**  
dplyr provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges. Go to docs...

**tidyr**



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, andforcats which you also may have seen before.

Model

Get help



## Import

As well as `readr`, for reading flat files, the `tidyverse` package installs a number of other packages for reading data:

- `DBI` for relational databases. (Maintained by Kirill Müller.) You'll need to pair `DBI` with a database specific backends like `RSQlite`, `RMariaDB`, `RPostgres`, or `odbc`. Learn more at <https://db.rstudio.com>.
- `haven` for SPSS, Stata, and SAS data.
- `httr` for web APIs.
- `readxl` for `.xls` and `.xlsx` sheets.
- `rvest` for web scraping.
- `jsonlite` for JSON. (Maintained by Jeroen Ooms.)
- `xml2` for XML.



## Wrangle

In addition to `tidyr` and `dplyr`, there are five packages (including `stringr` and `forcats`) which are designed to work with specific types of data:

## Wrangle

In addition to `tidyr`, and `dplyr`, there are five packages (including `stringr` and `forcats`) which are designed to work with specific types of data:

- `lubridate` for dates and date-times.
  - `date` for time-of-day values.
  - `blob` for storing blob (binary) data.

Wrangle

Program

Model



## Program

In addition to `purr`, which provides very consistent and natural methods for iterating on R objects, there are two additional `tidyverse` packages that help with general programming challenges:

- `magrittr` provides the pipe, `%>%` used throughout the `tidyverse`. It also provides a number of more specialised piping operators (like `%%%` and `%<>%`) that can be useful in other places.
- `glue` provides an alternative to `paste()` that makes it easier to combine data and strings.



## Model

Modeling with the `tidyverse` uses the collection of `tidymodels` packages, which largely replace

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

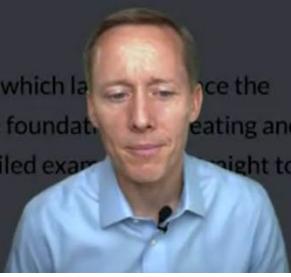
## Program

In addition to `purrr`, which provides very consistent and natural methods for iterating on R objects, there are two additional tidyverse packages that help with general programming challenges:

- `magrittr` provides the pipe, `%>%` used throughout the tidyverse. It also provides a number of more specialised piping operators (like `%%%` and `%<>%`) that can be useful in other places.
- `glue` provides an alternative to `paste()` that makes it easier to combine data and strings.

## Model

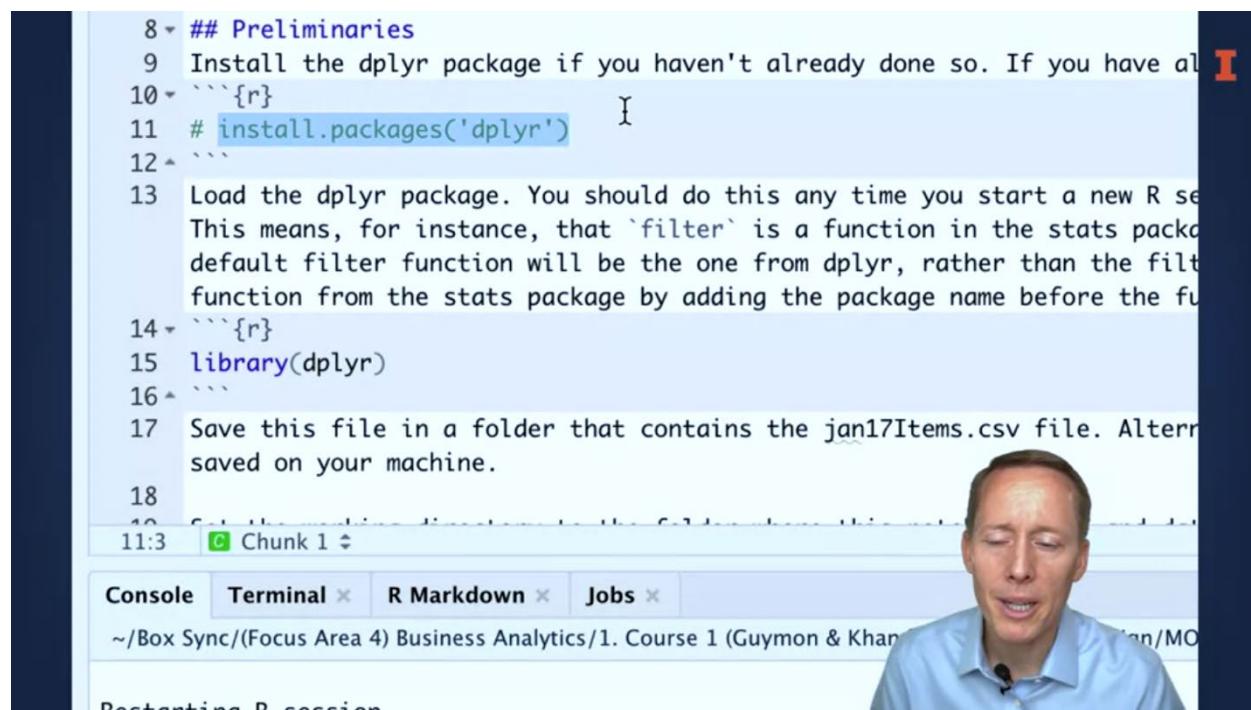
Modeling with the tidyverse uses the collection of `tidymodels` packages, which largely replace the `modelr` package used in `R4DS`. These packages provide a comprehensive foundation for creating and using models of all types. Visit the [Getting Started](#) guide or, for more detailed examples, right to the [Learn](#) page.



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.



The screenshot shows an RStudio interface. The code editor contains the following R code:

```
8 - ## Preliminaries
9 Install the dplyr package if you haven't already done so. If you have al I
10 - ``{r}
11 # install.packages('dplyr')    }
12 -
13 Load the dplyr package. You should do this any time you start a new R se I
This means, for instance, that `filter` is a function in the stats packa
default filter function will be the one from dplyr, rather than the fil
function from the stats package by adding the package name before the fu
14 - ``{r}
15 library(dplyr)
16 -
17 Save this file in a folder that contains the jan17Items.csv file. Alterr
saved on your machine.
18
19
11:3 C Chunk 1
```

The RStudio interface includes tabs for Console, Terminal, R Markdown, and Jobs. Below the tabs, the path is shown as ~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khan). A video player window is overlaid on the interface, showing a man in a blue shirt speaking. The video player has a progress bar and a timestamp of 0:00.

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.

Save this file in a folder that contains the jan17Items.csv file. Alternatively, move that file to the folder in which this notebook file is saved on your machine.

```

18
19 Set the working directory to the folder where this notebook file and data are saved. You can do that in one of at least two ways:
20
21 1. You can use the `setwd()` command, or
22 2. You can use the RStudio menu buttons: Session > Set Working Directory > To Source File Location
23
24 Read in the jan17Items data as j17i.
25 ~ ~(r)
26 j17i <- read.csv('jan17Items.csv')
27 str(j17i)
28 ...
29
30 ## Filtering Rows Using the Filter Function from the dplyr Package
31 You can filter rows of data to observations that meet a certain condition using the `filter()` function from the dplyr package along with the relational operator that pertains to your situation.
32 ~ ~(r)
33 highCost <- filter(j17i, Cost > 11)
34 dim(highCost)
35 ...
36
36.1 [green] Chunk 2 : R Markdown
```

Console Terminal R Markdown Jobs

Restarting R session...

```

> library(dplyr)
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':
```

File Explorer

Set Working Directory > To Source File Location

Name	Size	Modified
M4_06 NOTEBOOK REVIEW AND...	2.4 KB	DEC 17, 2020
.Rhistory	17.2 KB	Dec 2, 2020
M4_05 Data Structure Wide Ver...	1.3 KB	Dec 1, 2020
M4_04 Data Structure Control V...	2 KB	Dec 1, 2020
M4_14 Joining Data.nb.html	672.2 KB	Dec 1, 2020
M4_14 Joining Data.Rmd	6.6 KB	Dec 1, 2020
M4_13 Stacking and Sorting Da...	667.8 KB	Nov 30, 2020
M4_13 Stacking and Sorting Da...	5.4 KB	Nov 30, 2020
M4_12 Data Stacks.nb.html	655.2 KB	Nov 30, 2020
M4_12 Pivoting Dataframes Bet...	664.1 KB	Nov 30, 2020
M4_12 Pivoting Dataframes Bet...	3.8 KB	Nov 30, 2020
M4_12 Pivoting Data Between ...	654.8 KB	Nov 30, 2020
M4_11 Data Aggregation and S...	2.8 KB	Nov 30, 2020
M4_10 Handling Missing Values...	4 KB	Nov 28, 2020
M4_09 Using dplyr Mutate and ...	2.2 KB	Nov 28, 2020
M4_08 Useful Operators.Rmd	3.1 KB	Nov 28, 2020
jan17Items.csv	1.7 MB	Feb 20, 2020
jan17Weather.csv	1.5 KB	Feb 20, 2020
feb17Items.csv	1.6 MB	Feb 20, 2020
mar17Items.csv	1.8 MB	Feb 20, 2020

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.

The screenshot shows an RStudio interface with a video overlay of Professor Ronald Guymon. The RStudio console contains the following R code:

```

24 Read in the jan17Items data as j17i.
25 [r]
26 j17i <- read.csv("jan17Items.csv")
27 str(j17i)
28
29
30 ## Filtering Rows Using the Filter Function from the dplyr Package
31 You can filter rows of data to observations that meet a certain condition using the "filter()" function from the dplyr package along with the relational operator that pertains to your situation.
32 {r}
33 highCost <- filter(j17i, Cost > 11)
34 dim(highCost)
35
36
37 Compare this to the code to filter rows using base R.
38 {r}
39 highCostBase <- i17i[i17i$Cost > 11,]
16.1 [Chunk 2]

```

The R Markdown preview pane shows the following text:

Set the working directory to the folder where this notebook file and data are saved. You can do that in one of at least two ways:

1. You can use the `setwd()` command, or
2. You can use the RStudio menu buttons: Session > Set Working Directory > To Source File Location

At the bottom of the RStudio interface, it says "Restarting R session...".

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.

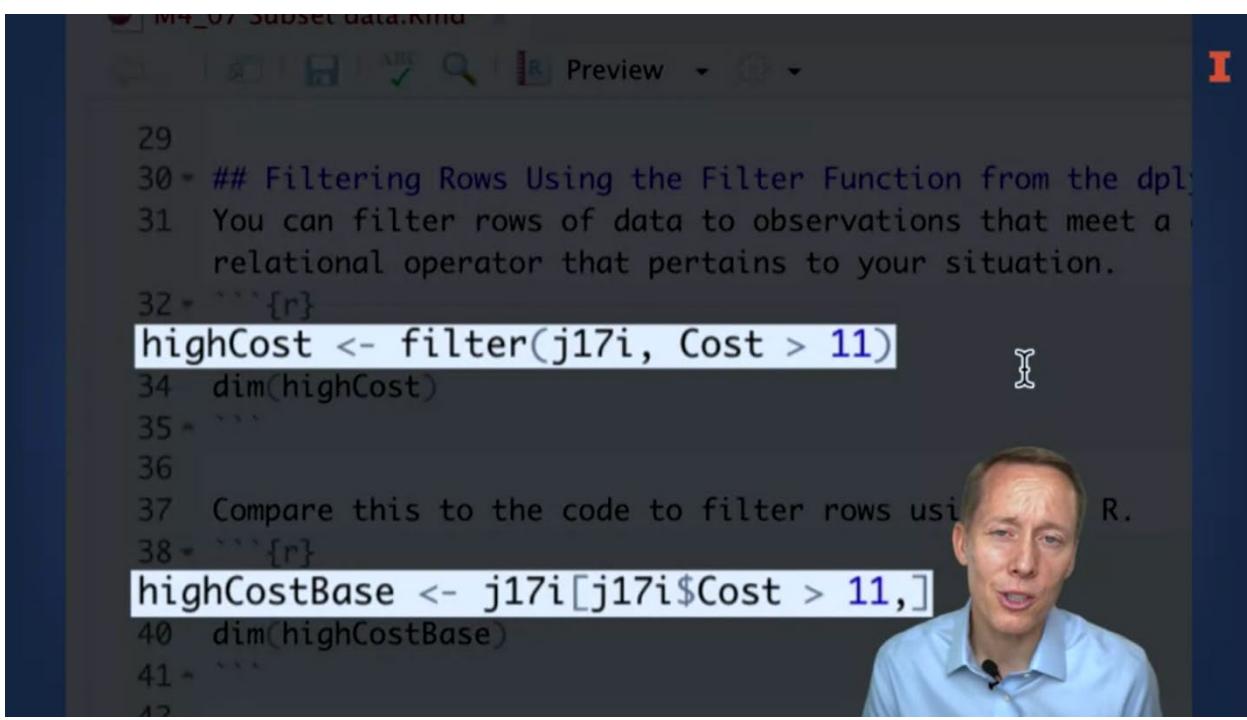
The screenshot shows the RStudio Session menu open, with the "Set Working Directory" option highlighted. The submenu shows three options: "To Source File Location" (which is selected), "To Files Pane Location", and "Choose Directory...". The RStudio interface shows the same R code and R Markdown preview as the previous screenshot.

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.

```
27 str(j17i)
28 ```

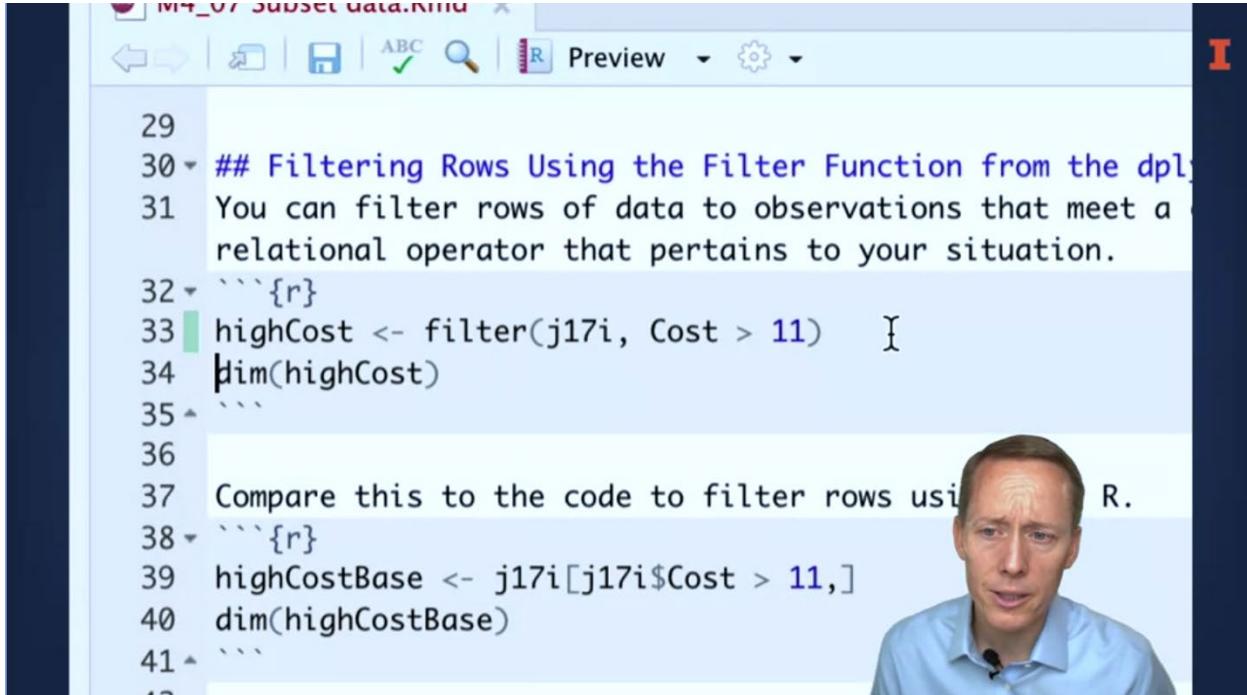
'data.frame': 8899 obs. of 21 variables:
 $ Time           : chr  "2017-01-26T21:18:00Z" "2017"
 $ OperationType : chr  "SALE" "SALE" "SALE" "SALE"
 $ BarCode         : chr  "*" "*" "*" "*" ...
 $ CashierName    : chr  "Nicholas Villines" "Carla K"
 $ LineItem        : chr  "Glass Mug" "Lamb Chops" "Sa"
 $ Department     : chr  "Beverage" "Entrees" "Entree"
 $ Category       : chr  "Glass Bottle" "Lamb Chops"
 $ CardholderName : chr  NA NA NA NA ...
 $ RegisterName   : chr  "RT149" "RT151" "RT151" "RT151"
 $ StoreNumber    : chr  "AZ23501411" "AZ23501411" "AZ23501411" "AZ23501411"
 $ TransactionNumber: chr  "00Z670S78157" "00Z670S78157" "00Z670S78157" "00Z670S78157"
 $ CustomerCode   : chr  "CWM11331L8" "CWM11331L8" "CWM11331L8" "CWM11331L8"
 $ Cost           : num  0 11 0 11
```

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.



M4\_07\_subset\_data.Rmd

```
29
30 ## Filtering Rows Using the Filter Function from the dplyr package
31 You can filter rows of data to observations that meet a
32 relational operator that pertains to your situation.
33 `##`{r}
34 highCost <- filter(j17i, Cost > 11)
35 dim(highCost)
36
37 Compare this to the code to filter rows using base R.
38 `##`{r}
39 highCostBase <- j17i[j17i$Cost > 11,]
40 dim(highCostBase)
41
42
```



M4\_07\_subset\_data.Rmd

```
29
30 ## Filtering Rows Using the Filter Function from the dplyr package
31 You can filter rows of data to observations that meet a
32 relational operator that pertains to your situation.
33 `##`{r}
34 highCost <- filter(j17i, Cost > 11)    #|
35 dim(highCost)
36
37 Compare this to the code to filter rows using base R.
38 `##`{r}
39 highCostBase <- j17i[j17i$Cost > 11,]
40 dim(highCostBase)
41
42
```

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.

The screenshot shows the RStudio interface. At the top, there's a navigation bar with tabs: Environment, History, Connections, Tutorial, and Presentation. Below the navigation bar, there are icons for file operations like Import Dataset and a search bar. The main area is titled "Global Environment". It lists two datasets: "highCost" (14 obs. of 21 variables) and "j17i" (8899 obs. of 21 variables). On the right side of the screen, there is a video overlay of a man in a blue shirt speaking.

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.

[1] 14 21

```

36
37 Compare this to the code to filter rows using base R.
38 - ``{r}
39 highCostBase <- j17i[j17i$Cost > 11]
40 dim(highCostBase)
41 -
42
43 Base R requires you to type the name of the dataframe twice.
44 functions, the tidyverse grammar is easier to read because it's less verbose in general.
45
46 You can also filter on multiple conditions. For instance,
47 if you wanted to find all rows where the price is greater than 13 and the cost
48 - ``{r}
49 highCostAndPrice <- filter(j17i, Cost > 13 & Price > 13)
```

Environment History Connections Tutorial Presentations

Import Dataset Global Environment

Data

highCost	14 obs. of 21 variables
highCostBase	14 obs. of 21 variables
j17i	8899 obs. of 21 variables

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

```
[1] 14 21

36
37 Compare this to the code to filter rows using base R.
38 ~`{r}
39 highCostBase <- j17i[j17i$Cost > 11,]
40 dim(highCostBase)
41 ~``

42
43 Base R requires you to type the name of the dataframe twice.
functions, the tidyverse grammar is easier to read because it's less verbose in general.
44
45
46 You can also filter on multiple conditions. For instance,
  if price is greater than 13 and quantity is less than 1000
47 ~`{r}
48 highCostBase <- j17i[j17i$Cost > 13 & j17i$Quantity <
```



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.

```
51  
52 If you want to be less explicit about which columns you want to keep, you can use the filter(.data, ..., .preserve = FALSE) argument.  
53 ````{r}  
54 highCostOrPrice <- filter(j17i, Cost > 11 | Price > 13)  
55 dim(highCostOrPrice)  
56 ````  
57  
58 ## Selecting Columns Using dplyr's Select Function  
59 You can reduce the width of a dataframe by selecting only certain columns. For example, if you would select only the Cost column:  
60 ````{r}  
61 highCost_Cost <- select(highCost, Cost)  
62 str(highCost_Cost)  
63 ````  
64
```





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.

```
51
52 If you want to be less exclusive and include observations w
53 ````{r}
54 highCostOrPrice <- filter(j17i, Cost > 11 | Price > 13)
55 dim(highCostOrPrice)
56 ````

[1] 3278   21     I

57 ---
58 ## Selecting Columns Using dplyr's Select Function
59 You can reduce the width of a dataframe by selecting only certain columns. If you would select only the Cost column:
60 ````{r}
61 highCost_Cost <- select(highCost, Cost)
56:1 C Chunk 7 ♦
```



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.

```

57
58 ## Selecting Columns Using dplyr's Select Function
59 You can reduce the width of a data frame by selecting only certain columns. If you would select only the Cost column:
60 `##[r]
61 highCost_Cost <- select(highCost, Cost)
62 str(highCost_Cost)
63 `##[r]

'data.frame':   14 obs. of  1 variable:
 $ Cost: num  19 19 189 189 189 ...

```

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.

```
$ Cost: num 19 19 189 189 189 ...

64
65 Notice that the highCost_Cost object is still a dataframe o
66 ````{r}
67 highCost_CostBase <- highCost[,c('Cost')]
68 str(highCost_CostBase)
69 ````

70
71 Base R is only a little more verbose, but it's definitely h
the dplyr approach in which the `select()` function explici
72
73 To select multiple columns, such as the Cost column and the colu
74 ````{r}
75 highCost_CostPrice <- select(highCost, Cost, P
76 str(highCost_CostPrice)
77 ````

78
```

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.

```
60 ````{r}
61 highCost_Cost <- select(highCost, Cost)
62 str(highCost_Cost)
63 ````

64
65 Notice that the highCost_Cost object is still a dataframe o
66 ````{r}
67 highCost_CostBase <- highCost[,c('Cost')]
68 str(highCost_CostBase)
69 ````

70
```

'data.frame': 14 obs. of 1 variable:  
 \$ Cost: num 19 19 189 189 189 ...

num [1:14] 19 19 189 189 189 ...



Select will retain the data frame object structure.

```
60 ~`{r}
61 highCost_Cost <- select(highCost, Cost)
62 str(highCost_Cost)
63 ~```

'data.frame':   14 obs. of  1 variable:
 $ Cost: num  19 19 189 189 189 ...

64
65 Notice that the highCost_Cost object is still a dataframe o
66 ~`{r}
highCost_CostBase <- highCost[,c('Cost')]
68 str(highCost_CostBase)
69 ~```

num [1:14] 19 19 189 189 189 ...

70
```



```
highCost_Cost <- select(highCost, Cost)
```

```
60 ~`{r}
61 highCost_Cost <- select(highCost, Cost)
62 str(highCost_Cost)
63 ~```

'data.frame':   14 obs. of  1 variable:
 $ Cost: num  19 19 189 189 189 ...

64
65 Notice that the highCost_Cost object is still a dataframe o
66 ~`{r}
67 highCost_CostBase <- highCost[,c('Cost')]
68 str(highCost_CostBase)
69 ~```

num [1:14] 19 19 189 189 189 ...

70
```



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.

```
71 Base R is only a little more verbose, but it's definitely h
    the dplyr approach in which the `select()` function explici
72
73 To select multiple columns, such as the Cost and Price colu
74 ````{r}
75 highCost_CostPrice <- select(highCost, Cost, Price)
76 str(highCost_CostPrice)
77 ````
```

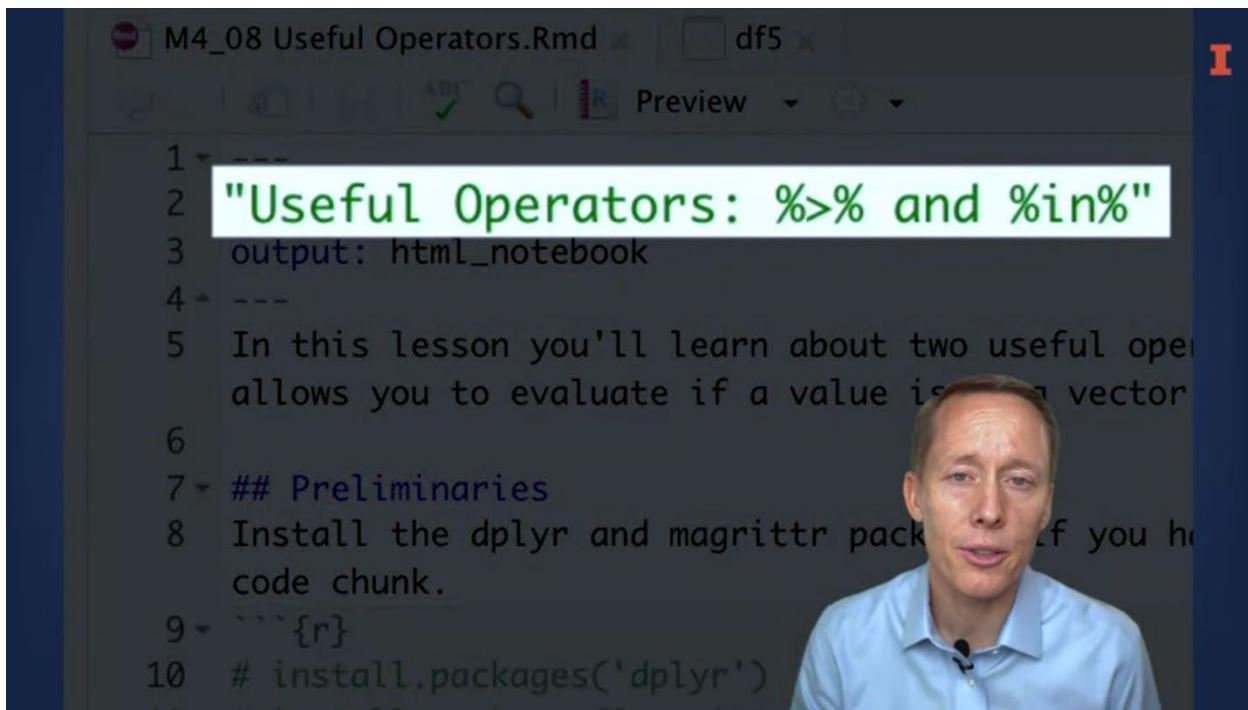
'data.frame': 14 obs. of 2 variables:

	Cost	Price
1	19	NaN
2	19	NaN
3	189	NaN
4	189	NaN
5	189	NaN
6	189	NaN
7	189	NaN
8	189	NaN
9	189	NaN
10	189	NaN
11	189	NaN
12	189	NaN
13	189	NaN
14	189	NaN

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%



The screenshot shows an RStudio interface with a code editor and a video player. The code editor contains an R Markdown file named 'M4\_08 Useful Operators.Rmd'. The title of the document is highlighted in green: **"Useful Operators: %>% and %in%"**. The code includes a section for preliminaries and a note about installing the dplyr package. A video of a man in a blue shirt is overlaid on the right side of the screen.

```
1 ---  
2 "Useful Operators: %>% and %in%"  
3 output: html_notebook  
4 ---  
5 In this lesson you'll learn about two useful operators.  
6 One allows you to evaluate if a value is in a vector.  
7 ## Preliminaries  
8 Install the dplyr and magrittr packages if you haven't done so.  
9 ````{r}  
10 # install.packages('dplyr')
```

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.

```
7 ## Preliminaries
8 Install the dplyr and magrittr packages if you have not done so.
9 ````{r}
10 # install.packages('dplyr')
11 # install.packages('magrittr')
12 ````

13 Load the dplyr and magrittr packages.
14 ````{r}
15 library(dplyr)
16 library(magrittr)
17 ````

18 Make sure that this file and the jan17Items.csv file are in the same folder.
19 ````
```



```
15 library(dplyr)
16 library(magrittr)
17 ````
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

```
18 Make sure that this file and the jan17Items.csv file are in the same folder.
19
20 Read in the jan17Items data as j17i.
```



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.



```
The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

Make sure that this file and the jan17Items.csv file are in the same folder and that the working directory is set to that folder.
19
20 Read in the jan17Items data as j17i.
21 +
22 j17i <- read.csv('jan17Items.csv')
23 +
24
25 ## Chaining Functions Together with the Pipe Operator: %>%
26 Assume that we want to filter the j17i dataframe to only the observations for which the value in the 'Cost' column is greater than 11, and
then select only the 'Cost' and 'Price' columns. There are at least two ways that you could do this.
27
28 First, you can create an intermediate dataframe.
29 +
30 df1 <- filter(j17i, Cost > 11)
31 df2 <- select(df1, Cost, Price)
32 +
33
15:1  [●] Chunk 2: R Markdown

Console Terminal R Markdown Jobs
~/Box Sync/[Focus Area 4] Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to process data?/rfiles/
> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':
  filter, lag

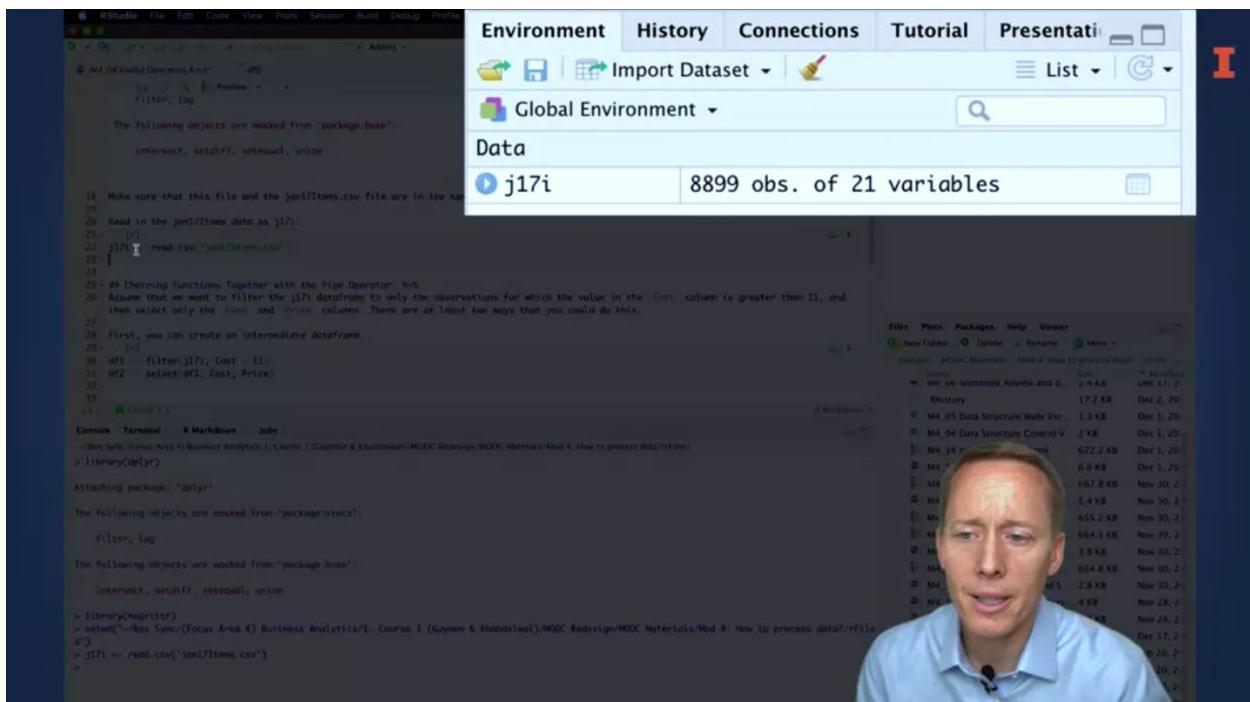
The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

libraryview[1:1]
```

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.

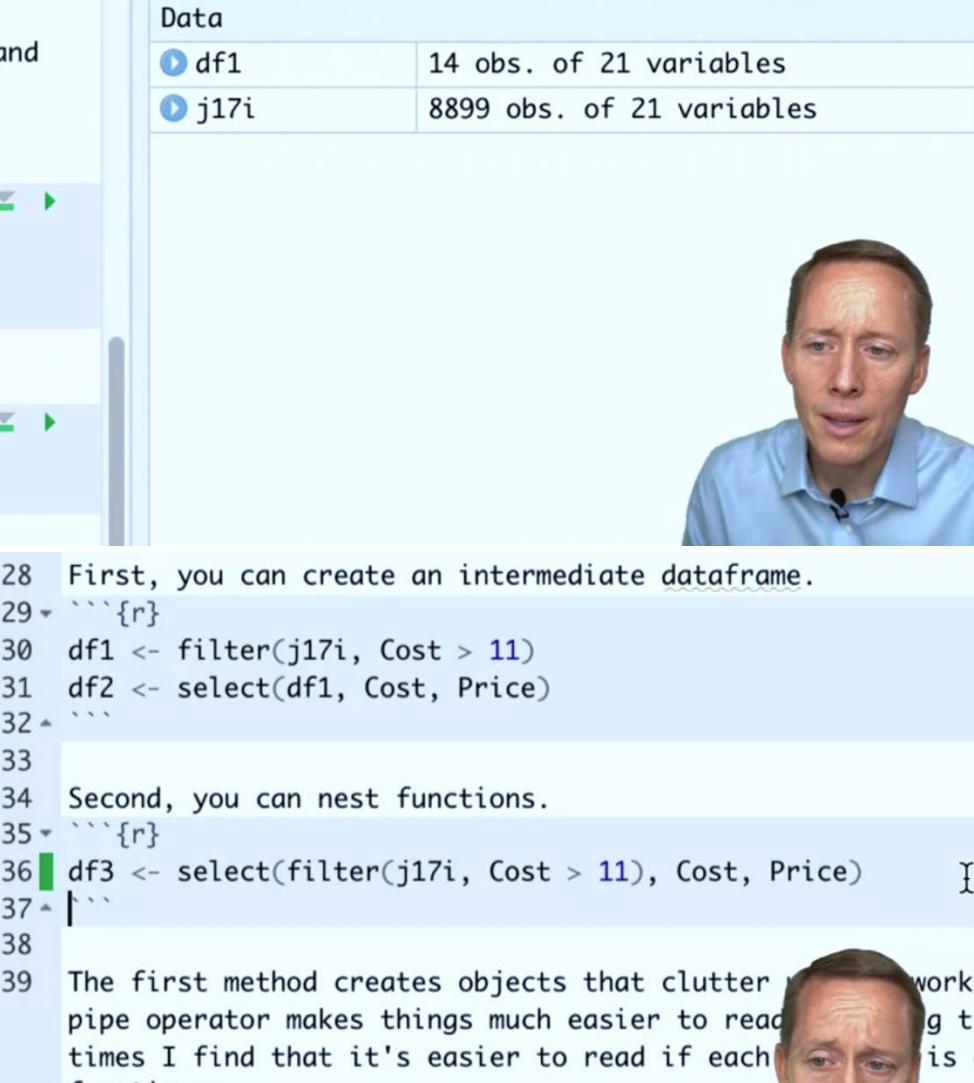


All right. Go ahead and Read in the Jan17Items.csv File and create a new Data Frame object called j17i.



```
M4_08 Useful Operators.Rmd* df5
24
## Chaining Functions Together with the Pipe Operator: %>%
25 Assume that we want to filter the j17i data frame to only the obs
26   then select only the `Cost` and `Price` columns. There are at le
27
28 First, you can create an intermediate data frame.
29 * ``{r}
30 df1 <- filter(j17i, Cost > 11)
31 df2 <- select(df1, Cost, Price)
32 * ``{
33
34 Second, you can nest functions
35 * ``{r}
36 df3 <- select(filter(j17i, Cost > 11), Cost,
37 * ``{
38
```

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.



L1, and

Environment History Connections Tutorial Presentations

Import Dataset

Global Environment

Data

df1	14 obs. of 21 variables
j17i	8899 obs. of 21 variables

First, you can create an intermediate dataframe.

```
28 First, you can create an intermediate dataframe.  
29 ````{r}  
30 df1 <- filter(j17i, Cost > 11)  
31 df2 <- select(df1, Cost, Price)  
32 ````  
33  
34 Second, you can nest functions.  
35 ````{r}  
36 df3 <- select(filter(j17i, Cost > 11), Cost, Price)  
37 ````  
38  
39 The first method creates objects that clutter up your working environment. The pipe operator makes things much easier to read. I think the pipe operator is on the out times I find that it's easier to read if each function is on a new line.
```

The first method creates objects that clutter up your working environment. The pipe operator makes things much easier to read. I think the pipe operator is on the out times I find that it's easier to read if each function is on a new line:

```
40 ````{r}  
41 df4 <- j17i %>%  
42 filter(Cost > 11) %>%  
43 select(Cost, Price)
```

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.

The screenshot shows the RStudio interface. The top menu bar includes Environment, History, Connections, Tutorial, and Presentation. Below the menu is a toolbar with icons for file operations like Import Dataset and a search bar. The Global Environment pane is open, displaying a list of objects:

Object	Description
df1	14 obs. of 21 variables
df2	14 obs. of 2 variables
df3	14 obs. of 2 variables
j17i	8899 obs. of 21 variables

A video overlay of a man speaking is visible in the bottom right corner of the RStudio window.

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.

```
27
28 First, you can create an intermediate dataframe.
29 + ````{r}
30 df1 <- filter(j17i, Cost > 11)
31 df2 <- select(df1, Cost, Price)
32 + ````{r}
33
34 Second, you can nest functions.
35 + ````{r}
df3 <- select(filter(j17i, Cost > 11), Cost, Price)````{r}
37 +
38
39 The first method creates objects that clutter up the environment. Working environments I find that it's easier to read if each function is on a new line. So I like to do this:
40 + ````{r}
41 df4 <- j17i %>%
42   filter(Cost > 11) %>%
```



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

```
times I find that it's easier to read if each function is on a new line. So I like to do this:
df4 <- j17i %>%
  filter(Cost > 11) %>%
  select(Cost, Price)
45 Notice that this is easy to write and read, and it's also clear what's needed and that end up cluttering the environment a lot.
46
47 ## Evaluating if an Element is in a Vector or Dataframe
```

44:1 [C] Chunk 6 ▾

Console Terminal × R Markdown × Jobs ×

~/Box Sync/(Focus Area 4) Business Analytics/1. Core Concepts and Techniques/1.1. Data Manipulation with R/

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.

```
47 ## Evaluating if an Element is in a Vector or Dataframe
48 Assume that we want to filter observations down to those for
function and manually type out a lot of OR statements, or you
49 ````{r}
50 df5 <- j17i %>%
51 filter(LineItemDescription %in% c('Glass Mug', 'Gift Cards'))
52 ````

53 This `%in%` operator can be especially handy especially if the
that we want to only look at observations for cardholders who
54 ````{r}
55 highCostItems <- j17i %>%
56 filter(Cost > 11)
57
58 custsThatPurchaseHighCostItems <- j17i %>%
59 filter(CardholderName %in% highCostItems$CardholderName)
60 ````

61 Consider how easy it would be to update the cost from 11 dollars.
```





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.

```

47 ## Evaluating if an Element is in a Vector or Dataframe
48 Assume that we want to filter observations down to those for which Lineitem is in a vector.
49 function and manually type out a lot of OR statements, or you could use the %in% operator.
50 df5 <- j17i %>%
51   filter(Lineitem %in% c('Glass Mug', 'Gift Cards'))
52
53 This `%in%` operator can be especially handy especially if there are many categories that we want to only look at observations for cardholders who purchase those items.
54 ``{r}
55 highCostItems <- j17i %>%
56   filter(Cost > 11)
57
58 custsThatPurchaseHighCostItems <- j17i %>%
59   filter(CardholderName %in% highCostItems$CardholderName)
60
61 Consider how easy it would be to update this code if we wanted to add more categories like gift cards or glass mugs.

```

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.

The screenshot shows the RStudio interface. On the left, there are two panes: 'Files' and 'Plots'. In the main area, the 'Data' browser is open, displaying a list of objects:

	Data
df1	14 obs. of 21 variables
df2	14 obs. of 2 variables
df3	14 obs. of 2 variables
df4	14 obs. of 2 variables
df5	98 obs. of 21 variables
j17i	8899 obs. of 21 variables

At the bottom, there is a navigation bar with tabs: Files, Plots, Packages, Help, and Viewer. Below the navigation bar are buttons for New Folder, Delete, and Rename.

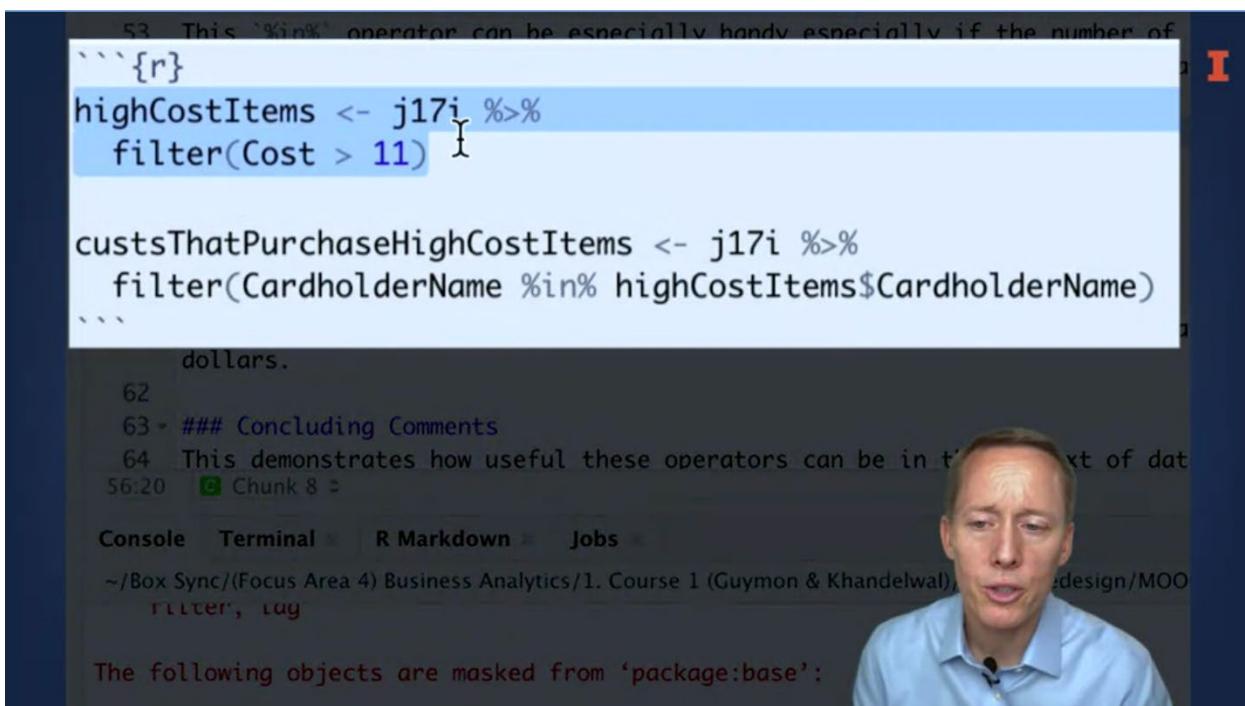
When I do that, if I run that, I've got this new DataFrame object that has 98 observations and 21 variables.



The screenshot shows the RStudio environment with two panes. The left pane displays a data frame named 'df5' with 98 rows and 21 columns. The columns include Time, OperationType, BarCode, CashierName, LineItem, Department, Category, and CardholderName. The 'LineItem' column contains values like 'Glass Mug', 'Gift Cards', and 'Gift Mug'. The right pane shows a subset of the data with columns Department, Category, and CardholderName. The 'Category' column includes entries such as 'Glass Bottle', 'Non Food', and 'Glass Bottle'. The 'CardholderName' column lists names like 'SHALAH DESERIO', 'ICESS SHIIBA', and 'ZACHARYAH MCTIGH'.

	Time	OperationType	BarCode	CashierName	LineItem	Department	Category	CardholderName
55	2017-01-17T12:28:00Z	SALE	*	Wallace Kuiper	Glass Mug	everage	Glass Bottle	SHALAH DESERIO
56	2017-01-16T13:52:00Z	SALE	*	Rachael Price	Glass Mug	everage	Glass Bottle	NA
57	2017-01-14T20:30:00Z	SALE	*	Nicholas Villines	Gift Cards	everage	Glass Bottle	NA
58	2017-01-14T17:24:00Z	SALE	*	Rachael Price	Glass Mug	Gift Cards	Non Food	ICESS SHIIBA
59	2017-01-14T15:41:00Z	SALE	*	Rachael Price	Glass Mug	everage	Glass Bottle	DELMAR SANDS
60	2017-01-14T12:07:00Z	SALE	*	Cheryl Williams	Glass Mug	everage	Glass Bottle	CORAN NICKLIEN
61	2017-01-13T19:21:00Z	SALE	*	Maryellen Smith	Glass Mug	everage	Glass Bottle	DOREMUS LANOIE
62	2017-01-13T18:42:00Z	SALE	*	Maryellen Smith	Glass Mug	everage	Glass Bottle	NAZARAH TUSTISON
63	2017-01-13T18:31:00Z	SALE	*	Maryellen Smith	Glass Mug	everage	Glass Bottle	NATAE UZZELL
64	2017-01-13T18:31:00Z	SALE	*	Maryellen Smith	Glass Mug	everage	Glass Bottle	NATAE UZZELL
65	2017-01-13T18:27:00Z	SALE	*	Maryellen Smith	Glass Mug	everage	Glass Bottle	SHOAN SAKASH
66	2017-01-13T16:44:00Z	SALE	*	Diana Miller	Glass Mug	everage	Glass Bottle	NA
67	2017-01-13T12:42:00Z	SALE	*	Cheryl Williams	Glass Mug	everage	Glass Bottle	RNELIA KESSEL
68	2017-01-12T11:47:00Z	SALE	*	Karen Castro	Glass Mug	everage	Glass Bottle	ETAL GOVERNALE
69	2017-01-12T11:41:00Z	SALE	*	Diana Miller	Glass Mug	everage	Glass Bottle	MARTEL NOACK
70	2017-01-12T11:29:00Z	SALE	*	Diana Miller	Glass Mug	everage	Glass Bottle	ZECHARYAH MCTIGH

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.



```
53 This %in% operator can be especially handy especially if the number of
```{r}
highCostItems <- j17i %>%
  filter(Cost > 11) ```

custsThatPurchaseHighCostItems <- j17i %>%
  filter(CardholderName %in% highCostItems$CardholderName)
```
dolars.
62
63 ## Concluding Comments
64 This demonstrates how useful these operators can be in the context of dat
56:20 [ ] Chunk 8 ▾
```

Console Terminal R Markdown Jobs

~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khandelwal)/BusinessAnalyticsDesign/MOO Filter, lug

The following objects are masked from ‘package:base’:

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.

| Data              |                           |
|-------------------|---------------------------|
| ⌚ custsThatPur... | 1760 obs. of 21 variables |
| ⌚ df1             | 14 obs. of 21 variables   |
| ⌚ df2             | 14 obs. of 2 variables    |
| ⌚ df3             | 14 obs. of 2 variables    |
| ⌚ df4             | 14 obs. of 2 variables    |
| ⌚ df5             | 98 obs. of 21 variables   |
| ⌚ highCostItems   | 20 obs. of 21 variables   |
| ⌚ j17i            | 8899 obs. of 21 variables |

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.

```
54 ````{r}
55 highCostItems <- j17i %>%
56   filter(Cost > 7)}
57
58 custsThatPurchaseHighCostItems <- j17i %>%
59   filter(CardholderName %in% highCostItems$CardholderName)
60 ``
61 Consider how easy it would be to update the observations if
dollars.
62
63 ## Concluding Comments
64 This demonstrates how useful these operators
```

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

"Using dplyr's Mutate, Rename, Relocate, and Distinct Functions"

```
5 This lesson focuses on four functions that simplify some common data preprocessing tasks: mutate(), rename(), relocate(), and distinct(). There are many other functions in the dplyr package for wrangling data. You should spend some time reviewing them when you want to perform a specific task.
6
7 ## Preliminaries
8 # Load the dplyr and magrittr packages.
9 #> #> (r)
10 library(dplyr)
11 library(magrittr)
12 ...
13 Make sure that this file and the jano17Items.csv file are in the same folder and that the working directory is set to that folder.
14
15 Read in the jano17Items.csv data as df1.
16 #> #> (r)
17 df1 <- read.csv('jano17Items.csv')
18 ...
19
20 ## Creating New Variables Using dplyr's Mutate Function
21 The 'mutate' function in the dplyr package makes it less verbose than using only base R by reducing the number of times that you have to type the name of the data frame.
22 ...
23
24 Chapt 3.R
25
```

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.

The video player shows a man in a light blue shirt speaking. To his right is a screenshot of a file browser window titled 'Mod 4: How to process data?/rFiles/'. The browser lists several files under the 'rFiles' folder, including 'jan17Items.csv'. The 'jan17Items.csv' file is highlighted.

| Name                              | Size     | Modified      |
|-----------------------------------|----------|---------------|
| M4_12 Pivoting Dataframes Bet...  | 665.1 KB | Dec 17, 20... |
| M4_12 Pivotin...                  | 3.8 KB   | Dec 17, 20... |
| M4_11 Data Aggregation and S...   | 661 KB   | Dec 17, 20... |
| M4_11 Data Aggregation and S...   | 2.8 KB   | Dec 17, 20... |
| M4_10 Handling Missing Values...  | 665.8 KB | Dec 17, 20... |
| M4_10 Handling Missing Values...  | 4.3 KB   | Dec 17, 20... |
| M4_08 Useful Operators.nb.html    | 661.9 KB | Dec 17, 20... |
| M4_08 Useful Operators.Rmd        | 3.1 KB   | Dec 17, 20... |
| M4_07 Subset data.nb.html         | 665.7 KB | Dec 17, 20... |
| M4_07 Subset data.Rmd             | 4.1 KB   | Dec 17, 20... |
| M4_06 Notebook Review and d...    | 659.5 KB | Dec 17, 20... |
| M4_06 Notebook Review and d...    | 2.4 KB   | Dec 17, 20... |
| M4_05 Data Structure Wide Ver...  | 1.3 KB   | Dec 1, 202... |
| M4_04 Data Structure Control V... | 2 KB     | Dec 1, 202... |
| M4_12 Data Stacks.nb.html         | 655.2 KB | Nov 30, 20... |
|                                   | 654.8 KB | Nov 30, 20... |
| feb17Items.csv                    | 1.7 MB   | Feb 20, 20... |
| mar17Items.csv                    | 1.5 KB   | Feb 20, 20... |
|                                   | 1.6 MB   | Feb 20, 20... |
|                                   | 1.8 MB   | Feb 20, 20... |

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.

The R code in the editor is:

```

16 ````{r}
17 j17i <- read.csv('jan17Items.csv')
18 ````
```

The data viewer window shows:

**Data**

j17i 8899 obs. of 21 variables

| Variables  | Type       | Format                 |
|------------|------------|------------------------|
| jan17Items | File       | jan17Items.csv         |
| jan17i     | data frame | 8899 rows × 21 columns |

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.

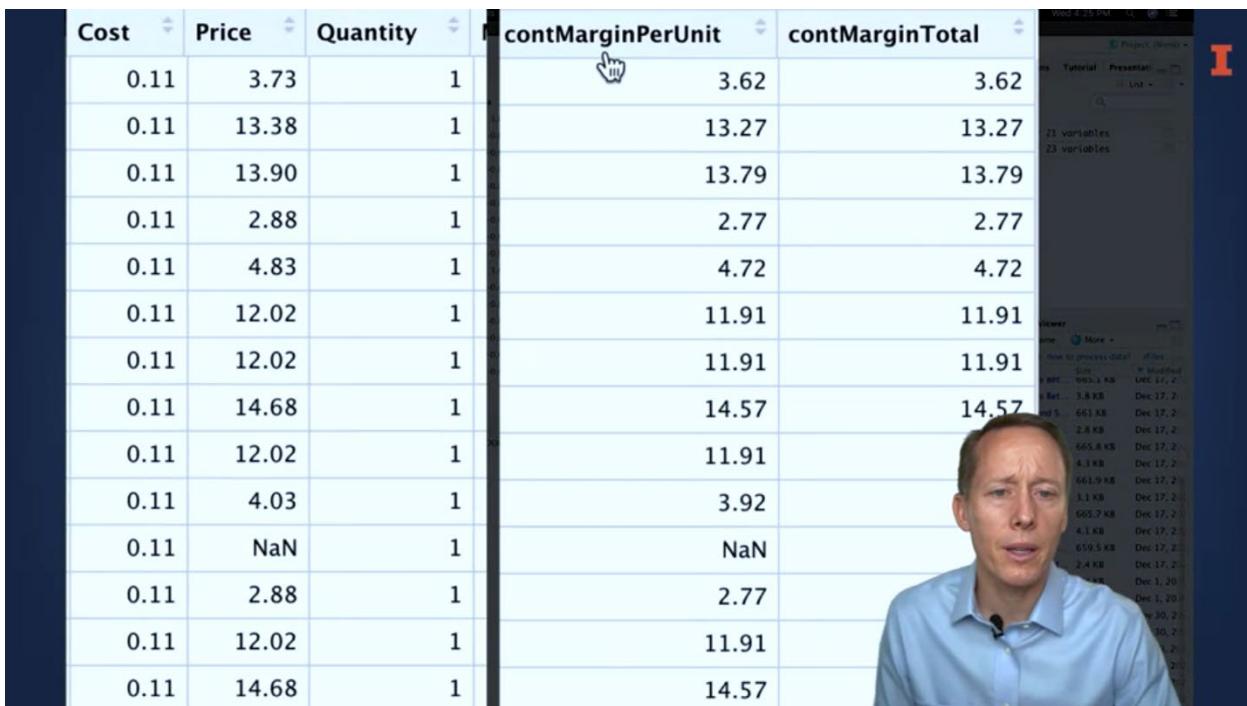
| Cost | Price | Quantity | Discounts | NetTotal | Tax | TotalOver |
|------|-------|----------|-----------|----------|-----|-----------|
| 0.11 | 3.73  | 1        |           |          |     |           |
| 0.11 | 13.38 | 1        |           |          |     |           |
| 0.11 | 13.90 | 1        |           |          |     |           |
| 0.11 | 2.88  | 1        |           |          |     |           |
| 0.11 | 4.83  | 1        |           |          |     |           |
| 0.11 | 12.02 | 1        |           |          |     |           |
| 0.11 | 4.03  | 1        |           |          |     |           |
| 0.11 | Nan   | 1        |           |          |     |           |
| 0.11 | 2.88  | 1        |           |          |     |           |
| 0.11 | 12.02 | 1        |           |          |     |           |
| 0.11 | 14.68 | 1        |           |          |     |           |
| 0.11 | 18.43 | 1        |           |          |     |           |
| 0.11 | 2.88  | 1        |           |          |     |           |

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.

```
25 - ``{r}
26   j17i_2 <- j17i %>%
27     mutate(
28       contMarginPerUnit = Price - Cost
29       , contMarginTotal = contMarginPerUnit * Quantity
30     )
31 - ``
32
33 To do the same thing with base R you would have to type
34 - ``{r}
35   j17i_2base <- j17i
36   j17i_2base$contMarginPerUnit <- j17i_2base$Price - j17i_2base$Cost
37   j17i_2base$contMarginTotal <- j17i_2base$contMarginPerUnit *
38 - ``
39
40 - ### Renaming Columns Using dplyr's R
```



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.



A screenshot of a video conference interface. On the left, there is a dark sidebar with the Illinois Gies College of Business logo. The main area shows a data frame in RStudio. The data frame has columns: Cost, Price, Quantity, contMarginPerUnit, and contMarginTotal. The contMarginPerUnit column contains values like 3.62, 13.27, 13.79, etc., with a cursor icon pointing to the first value. The contMarginTotal column contains values like 3.62, 13.27, 13.79, etc. To the right of the data frame, a video feed of a man in a blue shirt is visible, and the RStudio interface is partially visible in the background.

| 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  $3.73 - 0.11$ , 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 works. Now I can't do this in my head, but  $0.75 - 0.13$  is 0.62, and then multiplied by 36. That seems to be right, 22.32.

```

25+ `}`{r}
26 j17i_2 <- j17i %>%
27   mutate(
28     contMarginPerUnit = Price - Cost
29     , contMarginTotal = contMarginPerUnit * Quantity
30   )
31+
32
33 To do the same thing with base R you would have to type out the name of the dataframe much more, which
34+ `}`{r}
35 j17i_2base <- j17i
36 j17i_2base$contMarginPerUnit <- j17i_2base$Price - j17i_2base$Cost
37 j17i_2base$contMarginTotal <- j17i_2base$contMarginPerUnit * j17i_2base$Quantity
38+
39
40### Renaming Columns Using dplyr's Rename Function
41 Renaming columns is very easy to do using dplyr's `rename` function.
42
43+ `}`{r}
44 j17i_2 <- j17i_2 %>%
45   rename(cmu = contMarginPerUnit, cmt = contMarginTotal)
36:33 C Chunk 4 ↴

```

Console Terminal × Jobs ×

~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Redesign/Process data

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':



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.

```

The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

> library(magrittr)
> j17i_2 <- read.csv("jan17Items.csv")
> j17i_2 <- j17i_2 %>%
>   mutate(
>     `contMarginPerUnit` = Price - Cost
>   ,
>   `contMarginTotal` = contMarginPerUnit * Quantity
>   )
>
> View(j17i_2)
> j17i_2base <- j17i_2
> j17i_2base$contMarginPerUnit <- j17i_2base$Price - j17i_2base$Cost
> j17i_2base$contMarginTotal <- j17i_2base$contMarginPerUnit * j17i_2base$Quantity
> View(j17i_2base)

```

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.

```

40 ## Renaming Columns Using dplyr's Rename Function
41 Renaming columns is very easy to do using dplyr's `rename` function.
42
43 ```{r}
44 j17i_2 <- j17i_2 %>%
45   rename(cmu = contMarginPerUnit, cmt = contMarginTotal) I
46 ```
47
48 ## Rearranging Column Order Using dplyr's Relocate Function
49 Rearranging columns in a meaningful order can make analysis easier. For a dataframe, but it may make sense to place them next to other columns that are important for visual inspection. In modeling it's often helpful if the variables of interest are columns to the left. It can also make processes more efficient if you arrange columns to the left. Here are a couple of examples using dplyr.
50
51 ```{r}

```

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.

```

M4_09 Using dplyr Mutate and Dist
j17i_2      j17i
Filter
ber TransactionNumber CustomerCode Cost Price Quantity Modifiers Subtotal Discounts NetTotal Tax TotalDue cmu cmt
11 002670S78157 CWM11331L80 0.11 3.73 1 0.01 3.74 3.00 0.74 NaN NaN 3.62 3.62
51 00XT6G2179417 CXV10742CJW 0.11 13.38 1 0.01 13.39 -0.03 13.42 1.06 14.48 13.27 13.27
05 00XT6G2179417 CXV10742CJW 0.11 13.90 1 0.01 13.91 -0.03 13.94 1.09 15.03 13.79 13.79
89 00XT6G2179417 CXV10742CJW 0.11 2.88 1 0.01 2.89 -0.03 2.92 0.23 3.15 2.77 2.77
73 000U1FE123677 CWM11331L80 0.11 4.83 1 0.01 4.84 -0.03 4.87 0.38 5.25 4.72 4.72
67 000U1FE123677 CWM11331L80 0.11 12.02 1 2.36 14.38 -0.03 14.41 1.13 15.54 11.91 11.91
67 00PZKVD122494 CWM11331L80 0.11 12.02 1 2.36 14.38 -0.03 14.41 1.13 15.54 11.91 11.91
36 00ASRUQ151063 CWM11331L80 0.11 14.68 1 1.08 15.76 -0.03 15.79 1.23 17.02 14.57 14.57
67 00ASRUQ151063 CWM11331L80 0.11 12.02 1 1.08 13.10 -0.03 13.13 1.04 14.17 11.91 11.91
58 00MNUPU169118 CWM11331L80 0.11 4.03 1 0.01 4.04 3.63 0.41 NaN NaN 3.92 3.92
58 00MNUPU169118 CWM11331L80 0.11 NaN 1 0.01 26.33 -0.03 26.36 2.06 28.42 NaN NaN
88 00MNUPU169118 CWM11331L80 0.11 2.88 1 0.01 2.89 -0.03 2.92 0.23 3.15 2.77 2.77
67 00MNUPU169118 CWM11331L80 0.11 12.02 1 1.08 13.10 -0.03 13.13 1.04 14.17 11.91 11.91
36 00ASHWK44387 CWM11331L80 0.11 14.68 1 2.36 17.04 -0.03 17.07 1.34 18.41 14.57 14.57
33 00ERRTC154382 CWM11331L80 0.11 18.43 1 0.01 18.44 -0.03 18.47 1.45 19.92 18.32 18.32
81 00QA221233515 CWM11331L80 0.11 2.88 1 0.01 2.89 -0.03 2.92 0.23 2.77 2.77

Showing 1 to 16 of 8,899 entries, 25 total columns

Console Terminal Jobs
~/Box Sync/[Focus Area 4] Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to process data?/files/
The Following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

> library(magrittr)
> j17i <- read.csv('jan17Items.csv')
> j17i_2 <- j17i %>%
+   mutate(

```

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.

The RStudio interface on the right displays the following R code:

```

47
48. ## Rearranging Column Order Using dplyr's Relocate Function
49 Rearranging columns in a meaningful order can make analyses simpler. For instance, calculated columns are always added on to the far right of
50 a data frame, but it may make sense to place them next to the columns that the new columns are based on especially if you want to do a quick
51 visual inspection. In modeling it's often helpful if the variable also known as the dependent variable, is one of the farthest
52 columns to the left. It can also make processes more efficient
53 to type so that all of the numeric
54 columns are next to each other. Here are a couple of examples of
55 relocate.
56
57 relocate(cmty, cmt, .after = Quantity)
58
59
60
61 ## Identifying Distinct Values Using dplyr's Distinct Function
62 The distinct function returns only the unique values from a column.
63
64 Let's assume that once we've calculated the total contribution margin for each line item observation we want
65 which the contribution margin is greater than $40. However, we do not want to see repeat values of line item
66
67 # Rearranging Column Order Using dplyr's Relocate Function

```

The RStudio console shows:

```

Console Terminal > Jobs >
~/Box Sync/[Focus Area 4] Business Analytics/I: Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to process data/rFiles/
The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

> library(magrittr)
> j17i <- read.csv('jan17Items.csv')
> j17i_2 <- j17i %>%
+   mutate(

```

The help viewer for the relocate function shows the following documentation:

**relocate(.data, ..., .before = NULL, .after = NULL)**

**Value**

A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.

**tidy-select** Columns to move.

**before, after** **tidy-select** Destination of columns selected by .... Supplying neither will move columns to the left-hand side; specifying both is an error.

**Note**

Object of the same type as .data. The output has the following properties:

- Rows are not affected.
- The same columns appear in the output, but (usually) in a different place.

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.

```

48 - ### Rearranging Column Order Using dplyr's Relocate Function
49 Rearranging columns in a meaningful order can make analyses simpler. For instance, calculated columns are always
  a dataframe, but it may make sense to place them next to the columns that the new columns are based on especially
  visual inspection. In modeling it's often helpful if the variable of interest, also known as the dependent variable,
  columns to the left. It can also make processes more efficient if you arrange the columns based on data type so
  columns are next to each other. Here are a couple of examples of how dplyr's ?relocate function can make that
  easier.
50
51 j17i_2 <- j17i_2 %>%
52   relocate(cmu, cmt, .after = Quantity)
53
54
55 numFirst <- j17i_2 %>%
56   relocate(where(is.numeric), .before = where(is.character))
57
58
59
60
61 - ### Identifying Distinct Values Using dplyr's Distinct Function
62 The 'distinct' function returns only the unique values from a column.
63
64 Let's assume that once we've calculated the total contribution margin for each line item, we want a list of items
  which the contribution margin is greater than $40. However, we do not want to see repeated line items, only unique ones.
65
66 Q: How's how we would do that?
52:21  [1] Chunk 6: 

```

Console Terminal Jobs

~/Box Sync/[Focus Area 4] Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to process data/rFile.R

the following objects are masked from package:base :

intersect, setdiff, setequal, union

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.

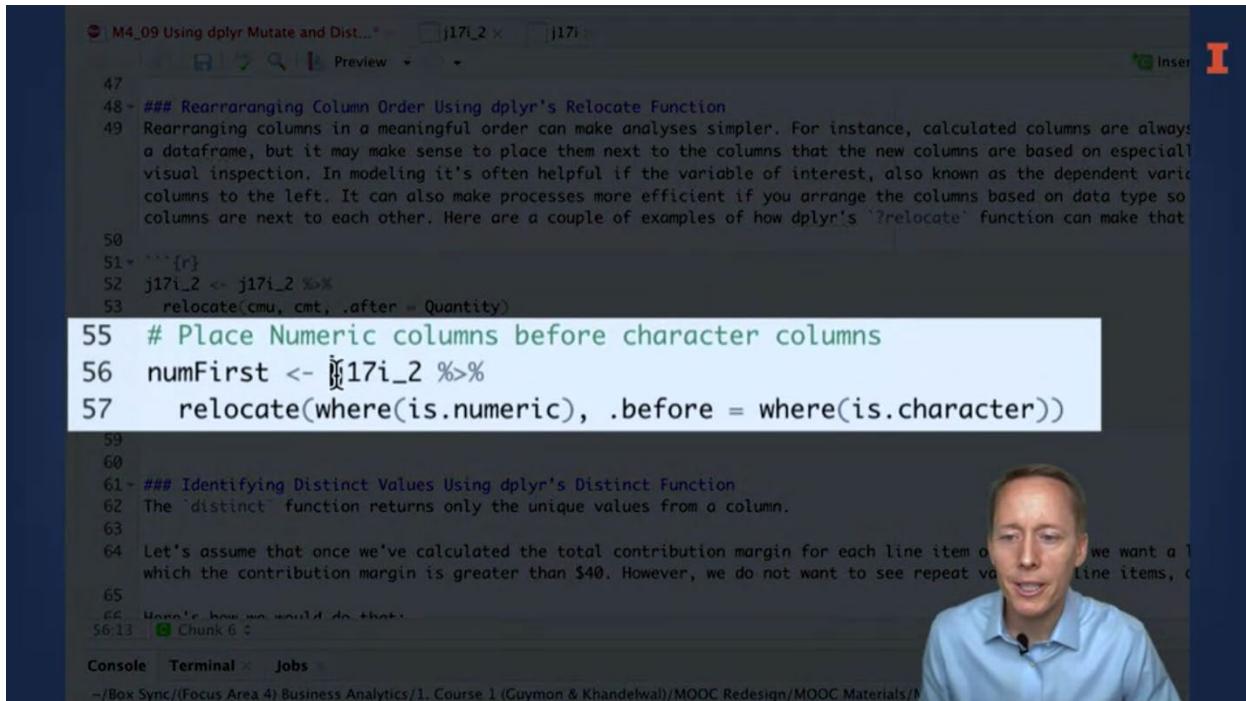
| Number | Cost        | Price | Quantity | cmu | cmt   | Discounts | NetTotal | Tax   |
|--------|-------------|-------|----------|-----|-------|-----------|----------|-------|
| 7      |             |       |          |     |       |           |          |       |
| 17     | CXV10742CJW | 0.11  | 13.38    | 1   | 13.27 | 13.27     | 0.01     | 13.39 |
| 17     | CXV10742CJW | 0.11  | 13.90    | 1   | 13.79 | 13.79     | 0.01     | 13.91 |
| 17     | CXV10742CJW | 0.11  | 2.88     | 1   | 2.77  | 2.77      | 0.01     | 2.89  |
| 77     | CWM11331L8O | 0.11  | 4.83     | 1   | 4.72  | 4.72      | 0.01     | 4.84  |
| 77     | CWM11331L8O | 0.11  | 12.02    | 1   | 11.91 | 11.91     | 2.36     | 14.38 |
| 94     | CWM11331L8O | 0.11  | 12.02    | 1   | 11.91 | 11.91     | 2.36     | 14.38 |
| 563    | CWM11331L8O | 0.11  | 14.68    | 1   | 14.57 | 14.57     | 1.08     | 15.76 |
| 563    | CWM11331L8O | 0.11  | 12.02    | 1   | 11.91 | 11.91     | 1.08     | 13.10 |
| 118    | CWM11331L8O | 0.11  | 4.03     | 1   | 3.92  | 3.92      | 0.01     | 4.04  |
| 118    | CWM11331L8O | 0.11  | NaN      | 1   | NaN   | NaN       | 0.01     | 26.33 |
| 118    | CWM11331L8O | 0.11  | 2.88     | 1   | 2.77  | 2.77      | 0.01     | 2.89  |
| 118    | CWM11331L8O | 0.11  | 12.02    | 1   | 11.91 | 11.91     | 1.08     | 13.10 |
| 87     | CWM11331L8O | 0.11  | 14.68    | 1   | 14.57 | 14.57     | 2.36     | 17.04 |
| 82     | CWM11331L8O | 0.11  | 18.43    | 1   | 18.32 | 18.32     | 0.01     | 18.44 |
| 15     | CWM11331L8O | 0.11  | 2.88     | 1   | 2.77  | 2.77      | 0.01     | 2.89  |

columns

Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to process data/rFile.R

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.



The screenshot shows a Jupyter Notebook interface with a dark theme. A video overlay of a man in a blue shirt is visible on the right side. The notebook cell contains the following R code:

```

47
48 ### Rearranging Column Order Using dplyr's Relocate Function
49 Rearranging columns in a meaningful order can make analyses simpler. For instance, calculated columns are always
50 a data frame, but it may make sense to place them next to the columns that the new columns are based on especially
51 visual inspection. In modeling it's often helpful if the variable of interest, also known as the dependent variable,
52 columns to the left. It can also make processes more efficient if you arrange the columns based on data type so
53 that columns are next to each other. Here are a couple of examples of how dplyr's 'relocate' function can make that
54 easier.
55 # Place Numeric columns before character columns
56 numFirst <- j17i_2 %>%
57   relocate(where(is.numeric), .before = where(is.character))
58
59
60
61 ### Identifying Distinct Values Using dplyr's Distinct Function
62 The 'distinct' function returns only the unique values from a column.
63
64 Let's assume that once we've calculated the total contribution margin for each line item or product, we want a list
65 of products that have a contribution margin greater than $40. However, we do not want to see repeat values for line items, o
66
67 numFirst %>% head(10) %>% dput()
68
69
```

Below the code cell, the status bar shows: "/Box Sync/[Focus Area 4] Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/M4\_09 Using dplyr Mutate and Dist...".

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.

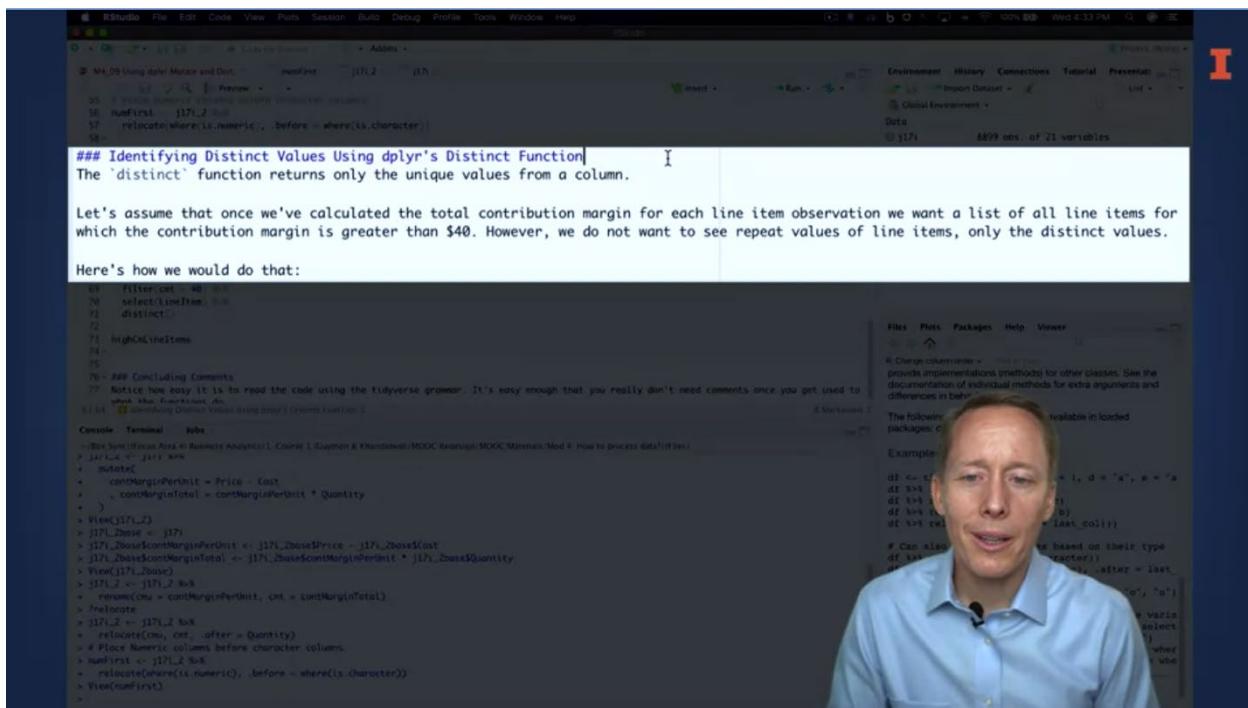
| Item                        | Department | Category                        | CardholderName     | RegisterName | StoreNumber | TransactionNumber | CustomerCode |
|-----------------------------|------------|---------------------------------|--------------------|--------------|-------------|-------------------|--------------|
| Mug                         | Beverage   | Glass Bottle                    | NA                 | RT149        | AZ23501411  | 002670578157      | CWM11331L80  |
| 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  |
| ain Drink                   | Beverage   | Fountain                        | NA                 | RT151        | AZ23501289  | 00XT6G2179417     | CXV10742CJW  |
| rgine and Chickpea Vindaloo | Entrees    | Aubergine and Chickpea Vindaloo | RAMSES LAQUAY      | RT151        | AZ23501473  | 00OU1FE123677     | CWM11331L80  |
| and Squash Kabob            | Kabobs     | Beef                            | RAMSES LAQUAY      | RT151        | AZ23501367  | 00OU1FE123677     | CWM11331L80  |
| and Squash Kabob            | Kabobs     | Beef                            | NAIYANNA RANTANEN  | RT151        | AZ23501367  | 00P2KVD122494     | CWM11331L80  |
| ken and Onion Kabob         | Kabobs     | Chicken                         | SAUDA COOKS        | RT151        | AZ23501336  | 00A5RUQ151663     | CWM11331L80  |
| and Squash Kabob            | Kabobs     | Beef                            | SAUDA COOKS        | RT151        | AZ23501367  | 00A5RUQ151663     | CWM11331L80  |
| and Apple Burgers           | Entrees    | Beef and Apple Burgers          | NA                 | RT149        | AZ23501258  | 00MNUPU169118     | CWM11331L80  |
| and Apple Burgers           | Entrees    | Beef and Apple Burgers          | NA                 | RT149        | AZ23501258  | 00MNUPU169118     | CWM11331L80  |
| i                           | Sides      | Naan                            | NA                 | RT149        | AZ23501388  | 00MNUPU169118     | CWM11331L80  |
| and Squash Kabob            | Kabobs     | Beef                            | NA                 | RT149        | AZ23501367  | 00MNUPU169118     | CWM11331L80  |
| ken and Onion Kabob         | Kabobs     | Chicken                         | CHAUNTEL MONA      | RT149        | AZ23501336  | 00A5HWK44387      | CWM11331L80  |
| on and Wheat Bran Salad     | Salad      | general                         | ANASTATIA SHELADIA | RT149        | AZ23501633  | 00ERRTC154382     | CWM11331L80  |
| ney                         | Sides      | Chutney                         | NA                 | RT149        | AZ23501381  | 00QIA22123515     | CWM11331L80  |

```

Showing 1 to 16 of 8,899 entries, 23 total columns
> View(j171_2)
> j171_2$base = j171_2
> j171_2$base$contMarginPerUnit <- j171_2$base$Price - j171_2$base$Cost
> j171_2$base$contMarginTotal <- j171_2$base$contMarginPerUnit * j171_2$base$Quantity
> View(j171_2$base)
> j171_2 <- j171_2 %>
> relocate(cmt = contMarginPerUnit, cmt = contMarginTotal)
> relocate(cmt = contMarginPerUnit, cmt = contMarginTotal)
> j171_2 <- j171_2 %>
> relocate(cmt, cmt, after = Quantity)
# Place Numeric columns before character columns
> numFirst <- j171_2 %>
> relocate(where(is.numeric), before = where(is.character))
> View(numFirst)
>

```

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.



The screenshot shows an RStudio interface with a dark theme. In the top-left pane, there is a code editor with the following R code:

```

## Identifying Distinct Values Using dplyr's Distinct Function
The 'distinct' function returns only the unique values from a column.

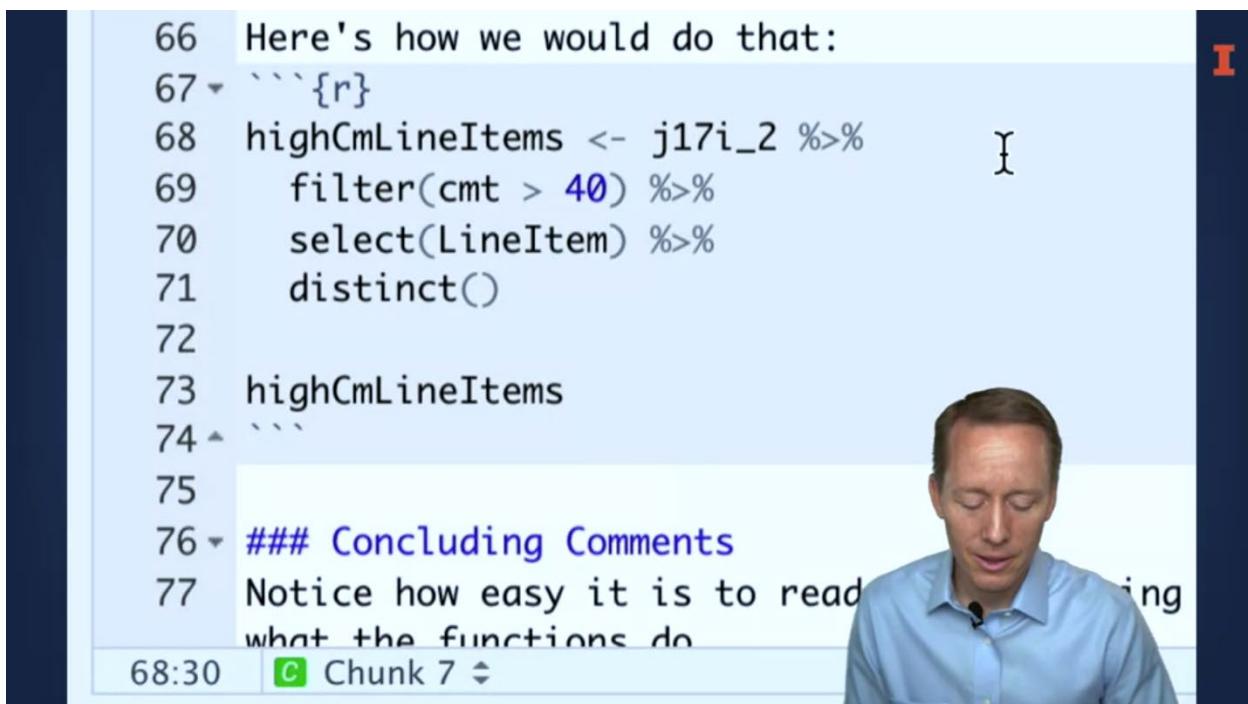
Let's assume that once we've calculated the total contribution margin for each line item observation we want a list of all line items for which the contribution margin is greater than $40. However, we do not want to see repeat values of line items, only the distinct values.

Here's how we would do that:

```

Below the code editor, the R console shows the execution of the code. The output includes the creation of a new data frame `highCmLineItems` and its contents.

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.



The screenshot shows the continuation of the R code from the previous slide. The code now includes the `filter`, `select`, and `distinct` functions, resulting in the creation of the `highCmLineItems` data frame.

```

66 Here's how we would do that:
67 ````{r}
68 highCmLineItems <- j17i_2 %>%
69   filter(cmt > 40) %>%
70   select(LineItem) %>%
71   distinct()
72
73 highCmLineItems
74 ````

75
76 ### Concluding Comments
77 Notice how easy it is to read and understand what the functions do

```

The bottom of the screen shows a timestamp "68:30" and a green "C" icon with "Chunk 7" below it.

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.

M4\_09 Using dplyr Mutate and Dist...\* numFirst j17i\_2 j17i

ABC Preview

72  
73 highCmLineItems  
74 ^`

**LineItem**  
<chr>

Chicken and Onion Kabob  
Beef and Broccoli  
Lamb Chops  
Beef and Squash Kabob  
Gift Cards  
Salmon and Wheat Bran Salad  
Aubergine and Chickpea Vindaloo  
Fountain Drink  
Beef and Broccoli Stir Fry

9 rows



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

```
13 Make sure that this file and the jan17Items.c
14
15 Read in the jan17Items data as j17i.
16 ````{r}
17 j17i <- read.csv('jan17Items.csv')
18 ````

19
20 ````{r}
21 ## Missing Values Review
22 NA and NAN values are treated differently than
23 that you want to remove NAs from your computat
24 ````{r}
25 v1 <- c(1,NA,3,9,15,NA)
26 sum(v1)
27 ````
```



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.

```

20 - ## Missing Values Review
21 NA and Nan values are treated differently than
   that you want to remove NAs from the computation
22 ````{r}
23 v1 <- c(1,NA,3,9,15,NA)
24 sum(v1, na.rm = T)
25 ````

[1] 28
26
27 Testing if a value is equal to NA is different
   than `==`.
28 ````{r}

```



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.

```

26
27 Testing if a value is equal to NA is different from test
   than `==`.
28 ````{r}
29 4 == NA # Returns NA
30 is.na(4) # Returns FALSE
31 use 'is.na' to check whether expression evaluates to
   NA

[1] NA
32
33
34 - ## Identifying the Pattern of Missing Values
35 When exploring a new dataset, it's worthwhile to identify
   values for each column along with the summary statistics
36 ````{r}
37 summary(j17i)

```



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.

VARIABLES FOR EACH COLUMN along with the summary statistics.

```
36 - ````(r)
37 summary(j17i)
38 - |
```

| Time             | OperationType    | BarCode          | CashierName      | LineItem         | Department       | Category         |
|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| Length:8899      |
| Class :character |
| Mode :character  |

| CardholderName   | RegisterName     | StoreNumber      | TransactionNumber | CustomerCode     | Cost             | Price            |
|------------------|------------------|------------------|-------------------|------------------|------------------|------------------|
| Length:8899      | Length:8899      | Length:8899      | Length:8899       | Length:8899      | Min. :-189.0400  | Min. :-322.810   |
| Class :character | Class :character | Class :character | Class :character  | Class :character | 1st Qu. : 0.1100 | 1st Qu. : 4.500  |
| Mode :character  | Mode :character  | Mode :character  | Mode :character   | Mode :character  | Median : 0.1100  | Median : 12.020  |
|                  |                  |                  |                   |                  | Mean : 0.2188    | Mean : 9.612     |
|                  |                  |                  |                   |                  | 3rd Qu. : 0.1100 | 3rd Qu. : 14.680 |
|                  |                  |                  |                   |                  | Max. : 189.0300  | Max. : 24.630    |
|                  |                  |                  |                   |                  | NA's : 562       |                  |

| Quantity         | Modifiers        | Subtotal        | Discounts         | NetTotal        | Tax             | TotalDue        |
|------------------|------------------|-----------------|-------------------|-----------------|-----------------|-----------------|
| Min. : 1.0000    | Min. :-2.5200    | Min. :-322.88   | Min. : -0.0300    | Min. :-322.77   | Min. :-25.340   | Min. :-348.11   |
| 1st Qu. : 1.0000 | 1st Qu. : 0.0100 | 1st Qu. : 5.24  | 1st Qu. : -0.0300 | 1st Qu. : 4.87  | 1st Qu. : 0.410 | 1st Qu. : 5.68  |
| Median : 1.0000  | Median : 0.0100  | Median : 13.10  | Median : -0.0300  | Median : 13.13  | Median : 1.040  | Median : 14.17  |
| Mean : 1.177     | Mean : 0.6548    | Mean : 15.10    | Mean : 0.7626     | Mean : 14.34    | Mean : 1.131    | Mean : 15.88    |
| 3rd Qu. : 1.0000 | 3rd Qu. : 1.0800 | 3rd Qu. : 15.76 | 3rd Qu. : -0.0300 | 3rd Qu. : 15.79 | 3rd Qu. : 1.230 | 3rd Qu. : 17.02 |
| Max. : 36.0000   | Max. : 57.5500   | Max. : 3328.12  | Max. : 951.8300   | Max. : 3328.15  | Max. : 261.260  | Max. : 3589.41  |
|                  |                  |                 |                   |                 | NA's : 341      | NA's : 341      |

39 Notice that the `Rate`, `Tax`, and `TotalDue` columns have missing values. The others numeric columns do not. Interesting that the number of missing values in `TotalDue` are the same. A visual inspection confirms that these are associated with the same observations.

40

41 - # Removing

42 If a column has a large percentage of missing values, then you may want to consider dropping that column all together. However, if you are really interested in the effect of that column, then a better idea is to remove the observations that have missing values. To indicate that you want to drop observations for which the value is NOT NA, then use the exclamation point to negate the `is.na()` function.

```
43 - ````(r)
44 j17i_2 <- j17i
45 filter(!is.na(j17i_2))
46 summary(j17i_2)
47 - |
38:1
```

R Markdown  

Console 

j17i  
Values  
v1 num

Files Plots Packages  
New Folder Delete  
redesign - MOOC Materials  
Name  
M4\_06 NOTEBOOK  
Rhistory  
M4\_05 Data Structures  
M4\_04 Data Structures  
M4\_14 Joining DataFrames  
M4\_14 Joining DataFrames  
M4\_13 Stacking  
M4\_13 Stacking  
M4\_12 Data Structures  
M4\_12 Data Structures  
M4\_12 Pivoting  
M4\_12 Pivoting  
M4\_12 Pivoting  
M4\_11 Data Aggregation  
M4\_10 Handling Data  
M4\_09 Using dplyr  
M4\_08 Useful Operations  
jan17Items.csv  
jan17Weather.csv  
feb17Items.csv  
mar17Items.csv

I

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.

| Line   | StoreNumber      | TransactionNumber | CustomerCode      | Cost             | Price           | Quantity       | Modifiers | Subtotal | Discounts | NetTotal | Tax   | TotalDue |
|--|------------------|-------------------|-------------------|------------------|-----------------|----------------|-----------|----------|-----------|----------|-------|----------|
| AZ23501258   | 00T99K472070     | CWM11331L80       | 0.11              | NaN              | 1               | 0.01           | 30.60     | 6.17     | 24.43     | 1.92     | 26.35 |          |
| AZ23501305   | 00T99K472070     | CWM11331L80       | 0.11              | NaN              | 1               | 0.01           | 29.52     | 5.85     | 23.67     | 1.86     | 25.53 |          |
| AZ23501350   | 0010KA62215      | CWM11331L80       | 0.11              | 13.38            | 1               | 1.08           | 14.46     | -0.03    | 14.49     | 1.13     | 15.62 |          |
| AZ23501404   | 00FW2B174660     | CWM11331L80       | 0.11              | 2.88             | 1               | 0.01           | 2.89      | -0.03    | 2.92      | 0.23     | 3.15  |          |
| AZ23501258   | 00CGMV4154080    | CWM11331L80       | 0.11              | 2.27             | 1               | 0.01           | 2.28      | 1.46     | 0.82      | NaN      | NaN   |          |
| AZ23501258   | 00CGMV4154080    | CWM11331L80       | 0.11              | 11.52            | 1               | 0.01           | 11.53     | -0.03    | 11.56     | 0.90     | 12.46 |          |
| AZ23501305   | 00CGMV4154080    | CWM11331L80       | 0.11              | 13.13            | 1               | 0.01           | 13.14     | -0.03    | 13.17     | 1.04     | 14.21 |          |
| AZ23501411   | 00CGMV4154080    | CWM11331L80       | 0.11              | 3.73             | 1               | 0.01           | 3.74      | -0.03    | 3.77      | 0.30     | 4.07  |          |
| AZ23501367   | 00CGMV4154080    | CWM11331L80       | 0.11              | 12.02            | 1               | 1.08           | 13.10     | -0.03    | 13.13     | 1.04     | 14.17 |          |
| AZ23501305   | 00K8YOU172818    | CWM11331L80       | 0.11              | 17.43            | 1               | 0.01           | 17.44     | -0.03    | 17.47     | 1.37     | 18.84 |          |
| AZ23501305   | 00K8YOU172818    | CWM11331L80       | 0.11              | 8.71             | 1               | 0.01           | 8.72      | -0.03    | 8.75      | 0.69     | 9.44  |          |
| AZ23501633   | 00K8YOU172818    | CWM11331L80       | 0.11              | 18.43            | 1               | 0.60           | 19.03     | -0.03    | 19.06     | 1.50     | 20.56 |          |
| AZ23501305   | 00K8YOU172818    | CWM11331L80       | 0.11              | 14.72            | 1               | 0.01           | 14.73     | -0.03    | 14.76     | 1.16     | 15.92 |          |
| AZ23501596   | 00K8YOU172818    | CWM11331L80       | 0.11              | 4.50             | 1               | 0.01           | 4.51      | -0.03    | 4.54      | 0.35     | 4.89  |          |
| AZ23501305   | 00C4WNS114516    | CWM11331L80       | 0.11              | NaN              | 1               | 0.01           | 29.10     | -0.03    | 29.13     | 2.29     | 31.42 |          |
| AZ23501411   | 00C4WNS114516    | CWM11331L80       | 0.11              | 3.73             | 1               | 0.01           | 3.74      | -0.03    | 3.77      | 0.30     | 4.07  |          |
| Showing 30 to 55 of 8,800 entries 21 total columns |                  |                   |                   |                  |                 |                |           |          |           |          |       |          |
| CardholderName                                     | RegisterName     | StoreNumber       | TransactionNumber | CustomerCode     | Cost            | Price          |           |          |           |          |       |          |
| Length:899   | Length:899       | Length:899        | Length:899        | Length:899       | Min.: -189.8400 | Min.: -322.810 |           |          |           |          |       |          |
| Class: character                                   | Class: character | Class: character  | Class: character  | Class: character | 1st Qu.: 0.1500 | 1st Qu.: 4.500 |           |          |           |          |       |          |
| Mode: character                                    | Mode: character  | Mode: character   | Mode: character   | Mode: character  | Median: 0.1100  | Median: 12.020 |           |          |           |          |       |          |
| Mean: 0.1777                                       | Mean: 0.6543     | Mean: 15.18       | Mean: 0.7625      | Mean: 1.34       | Mean: 0.1200    | Mean: 9.632    |           |          |           |          |       |          |
| 3rd Qu.: 1.0000                                    | 3rd Qu.: 0.8800  | 3rd Qu.: 15.76    | 3rd Qu.: 0.8300   | 3rd Qu.: 15.79   | 3rd Qu.: 0.2300 | 3rd Qu.: 17.07 |           |          |           |          |       |          |
| Max.: 36.0000                                      | Max.: 37.5500    | Max.: 3328.12     | Max.: 951.8300    | Max.: 5328.19    | Max.: 261.260   | Max.: 3589.41  |           |          |           |          |       |          |
| Quantity   | Modifiers        | Subtotal          | Discounts         | NetTotal         | Tax             | TotalDue       |           |          |           |          |       |          |
| Min.: 1.0000                                       | Min.: -2.5200    | Min.: -322.80     | Min.: -0.8300     | Min.: -322.77    | Min.: -25.340   | Min.: -348.11  |           |          |           |          |       |          |
| 1st Qu.: 1.0000                                    | 1st Qu.: 0.0100  | 1st Qu.: 3.24     | 1st Qu.: -0.0100  | 1st Qu.: 4.67    | 1st Qu.: 0.410  | 1st Qu.: 5.68  |           |          |           |          |       |          |
| Median: 1.0000                                     | Median: 0.0200   | Median: 15.18     | Median: -0.3500   | Median: 13.13    | Median: 1.940   | Median: 14.17  |           |          |           |          |       |          |
| Mean: 1.1777                                       | Mean: 0.6543     | Mean: 15.18       | Mean: 0.7625      | Mean: 1.34       | Mean: 1.130     | Mean: 15.88    |           |          |           |          |       |          |
| 3rd Qu.: 1.0000                                    | 3rd Qu.: 0.8800  | 3rd Qu.: 15.76    | 3rd Qu.: 0.8300   | 3rd Qu.: 15.79   | 3rd Qu.: 0.2300 | 3rd Qu.: 17.07 |           |          |           |          |       |          |
| Max.: 36.0000                                      | Max.: 37.5500    | Max.: 3328.12     | Max.: 951.8300    | Max.: 5328.19    | Max.: 261.260   | Max.: 3589.41  |           |          |           |          |       |          |
|  |                  |                   |                   |                  | NA's: 341       | NA's: 341      |           |          |           |          |       |          |

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.



The RStudio interface shows the following code in the console:

```

```{r}
j17i_2 <- j17i %>%
  filter(!is.na(Price))
summary(j17i_2)
```

```

The output of the `summary` command is displayed below:

|                 | CustomerCode     | Cost           | Price |
|-----------------|------------------|----------------|-------|
| Min. :-189.0400 | Min. :-322.810   | Min. :-348.11  |       |
| 1st Qu.: 0.1180 | 1st Qu.: 4.500   | 1st Qu.: 5.66  |       |
| Median : 0.0100 | Median : -0.0300 | Median : 14.17 |       |
| Mean : 0.0548   | Mean : 0.7626    | Mean : 15.86   |       |
| 3rd Qu.: 1.0000 | 3rd Qu.: 15.76   | 3rd Qu.: 17.87 |       |
| Max. :36.0000   | Max. :57.5500    | Max. :261.260  |       |

Below this, there is a table for the `j17i` dataframe:

|                 | CustomerCode     | Cost           | Price |
|-----------------|------------------|----------------|-------|
| Min. :-189.0400 | Min. :-322.810   | Min. :-348.11  |       |
| 1st Qu.: 0.1180 | 1st Qu.: 4.500   | 1st Qu.: 5.66  |       |
| Median : 0.0100 | Median : -0.0300 | Median : 14.17 |       |
| Mean : 0.0548   | Mean : 0.7626    | Mean : 15.86   |       |
| 3rd Qu.: 1.0000 | 3rd Qu.: 15.76   | 3rd Qu.: 17.87 |       |
| Max. :36.0000   | Max. :57.5500    | Max. :261.260  |       |

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.

The video shows a screen recording of RStudio. In the Global Environment pane, there are two dataframes: `j17i` (8899 obs. of 21 variables) and `j17i_2` (8337 obs. of 21 variables). The `j17i_2` entry has a small red arrow pointing to it. The code editor shows a script named `M4_10 Handling Missing Values.Rmd` with several lines of R code. The console output shows the creation of `j17i_2` from `j17i` by filtering out rows where the `Price` column is missing. The data summary for `j17i_2` shows that all columns have 8337 non-NA observations.

```

j17i_2 <- j17i %>
  filter(!is.na(Price))
  summary(j17i_2)

```

The video player interface at the bottom indicates the video is at 1:20 of a 10:20 duration.

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.

```

## Imputing Missing Values with the ifelse Function
# If you don't want to lose the other information from the incomplete observation, then you can fill in the missing values with the mean or median value from that column. To do that, you can use the `ifelse()` function, which is identical to the `=IFC()` function in Excel. We'll do that within the mutate function, and then calculate the value for TotalDue.
60:   [x]
61:   ifelse
62:   #> #> 3275 NA
63:   mutate
64:   #> #> Tax = ifelse(is.na(Tax), mean(Tax, na.rm = TRUE), Tax) # Could also substitute instead of mean(NA) then calculate
65:   #> #> , TotalDue = NetTotal + Tax
66:   #> #>
67:   summary(Jobs)
68:   [x]
69:
70: The `ifelse` function is powerful, and you can use it in many other ways. Conditional statements will be covered elsewhere in more depth.
71:
72: There are other ways for dealing with missing values. A model-based approach fills in missing values based on values from one or more other columns.
73:
74: ## Concluding Comments
75: You can see that the easiest approach for dealing with missing values is to remove the columns or the observations; however, you may be throwing the baby out with the bathwater, especially with small datasets. Ultimately, you'll have to use judgment for dealing with missing values. The important thing is to fully disclose what you do.
76: 
```

Console Terminal - R Markdown - Jobs

```

Price Quantity Modifiers Subtotal Discounts NetTotal Tax
Min. : 322.80 Min. :1.000 Min. :2.5200 Min. :322.80 Min. :322.77 Min. :25.3400
1st Qu.: 4.500 1st Qu.:1.000 1st Qu.: 0.0100 1st Qu.: 4.84 1st Qu.: -0.0300 1st Qu.: 4.54 1st Qu.: 0.4000
Median : 12.020 Median :1.000 Median : 0.0100 Median : 12.61 Median : -0.0300 Median : 12.06 Median : 1.0000
Mean : 9.612 Mean : 1.236 Mean : 0.0982 Mean : 10.62 Mean : -0.3395 Mean : 10.82 Mean : 0.8769
3rd Qu.: 34.680 3rd Qu.:3.000 3rd Qu.: 0.8800 3rd Qu.: 34.68 3rd Qu.: -0.8900 3rd Qu.: 34.58 3rd Qu.: 3.2200
Max. : 24.630 Max. :30.000 Max. :57.5500 Max. :396.1100 Max. : 232.35 Max. : 39.6100
NAs: 328

```

```

TotalDue
Min. : 322.80
1st Qu.: 5.50
Median : 13.70
Mean : 12.87
3rd Qu.: 16.89
Max. : 228.96
NA's: 328

```

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.

```

If you don't want to lose the other information from the incomplete observation, then you can fill in the missing values with the mean or median value from that column. To do that, you can use the `ifelse()` function, which is identical to the `=IFC()` function in Excel. We'll do that within the mutate function, and then calculate the value for TotalDue.
60:   ifelse
61:   #> #> 3275 NA
62:   mutate
63:   #> #> Tax = ifelse(is.na(Tax), mean(Tax, na.rm = TRUE), Tax) # Could also substitute instead of mean(NA) then calculate
64:   #> #> , TotalDue = NetTotal + Tax
65:   #> #>
66:   summary(Jobs)
67:   [x]
68:
69:
70: The `ifelse` function is powerful, and you can use it in many other ways. Conditional statements will be covered elsewhere in more depth.
71:
72: There are other ways for dealing with missing values. A model-based approach fills in missing values based on values from one or more other columns.
73:
74: ## Concluding Comments
75: You can see that the easiest approach for dealing with missing values is to remove the columns or the observations; however, you may be throwing the baby out with the bathwater, especially with small datasets. Ultimately, you'll have to use judgment for dealing with missing values. The important thing is to fully disclose what you do.
76: 
```

Console Terminal - R Markdown - Jobs

```

Price Quantity Modifiers Subtotal Discounts NetTotal Tax
Min. : 322.80 Min. :1.000 Min. :2.5200 Min. :322.80 Min. :322.77 Min. :25.3400
1st Qu.: 4.500 1st Qu.:1.000 1st Qu.: 0.0100 1st Qu.: 4.84 1st Qu.: -0.0300 1st Qu.: 4.54 1st Qu.: 0.4000
Median : 12.020 Median :1.000 Median : 0.0100 Median : 12.61 Median : -0.0300 Median : 12.06 Median : 1.0000
Mean : 9.612 Mean : 1.236 Mean : 0.0982 Mean : 10.62 Mean : -0.3395 Mean : 10.82 Mean : 0.8769
3rd Qu.: 34.680 3rd Qu.:3.000 3rd Qu.: 0.8800 3rd Qu.: 34.68 3rd Qu.: -0.8900 3rd Qu.: 34.58 3rd Qu.: 3.2200
Max. : 24.630 Max. :30.000 Max. :57.5500 Max. :396.1100 Max. : 232.35 Max. : 39.6100
NAs: 328

```

```

TotalDue
Min. : 322.80
1st Qu.: 5.50
Median : 13.70
Mean : 12.87
3rd Qu.: 16.89
Max. : 228.96
NA's: 328

```

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.

```

median value from that column. To do that, you can use the `ifelse()` function, which
that within the mutate function, and then calculate the value for TotalDue.

60+ ```{r}
61  ?ifelse
62  j17i_3 <- j17i %>%
63    mutate(
64      Tax = ifelse(is.na(Tax), mean(Tax, na.rm = T), Tax) # Could use median() instead of mean()
65      , TotalDue = NetTotal + Tax
66    )
67  summary(j17i_3)
68+ ````
69
70 The `ifelse` function is powerful, and you can use it in many other ways. Conditional
71
72 There are other ways for dealing with missing values. A model-based approach fills in
columns.
73
74+ ## Concluding Comments
75 You can see that the easiest approach for dealing with missing values is to remove them
throwing the baby out with the bathwater, especially with small datasets. Ultimately,
values. The important thing is to fully disclose what you do.

```

64:52 [C] Chunk 8 ↴

Console Terminal × R Markdown × Jobs ×

~/Box Sync//Focus Area 4/Business Analytics/1. Course 1 (Guymon & Khandelwal)



So to use that in this j17i data frame, 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.

```

58 - |{|
59 | ifelse
60 | j17L3 -- j17L3|}|
61 | mutate|
62 |   Tax = ifelse(is.na.Tax), mean(Tax, na.rm = T), Tax) # Could use median() instead of mean() if there are outliers
63 |   , TotalDue = NetTotal + Tax|
64 | }|
65 | summary(j17L3)
66 |
67 |
```

|          | Cost      | Price           |
|----------|-----------|-----------------|
| Min.     | -189.0400 | Min. : -322.810 |
| 1st Qu.: | 0.1100    | 1st Qu.: 4.500  |
| Median : | 0.1100    | Median : 12.020 |
| Mean :   | 0.2108    | Mean : 9.612    |
| 3rd Qu.: | 0.1100    | 3rd Qu.: 14.680 |
| Max. :   | 189.0300  | Max. : 24.630   |
| NA's     |           | : 562           |

|          | Tax     | TotalDue       |
|----------|---------|----------------|
| Min.     | -25.340 | Min. : -348.11 |
| 1st Qu.: | 0.460   | 1st Qu.: 5.25  |
| Median : | 1.040   | Median : 14.17 |
| Mean :   | 1.131   | Mean : 15.47   |
| 3rd Qu.: | 1.230   | 3rd Qu.: 17.02 |
| Max. :   | 261.260 | Max. : 3589.41 |

|          | Quantity | Modifiers       | Subtotal       | Discounts        | Net            |
|----------|----------|-----------------|----------------|------------------|----------------|
| Min. :   | 1.000    | Min. : -2.5200  | Min. : -322.80 | Min. : -0.0300   | Min. : -322.81 |
| 1st Qu.: | 1.000    | 1st Qu.: 0.0100 | 1st Qu.: 5.24  | 1st Qu.: -0.0300 | 1st Qu.: 4.50  |
| Median : | 1.000    | Median : 0.0100 | Median : 13.10 | Median : -0.0300 | Median : 12.02 |
| Mean :   | 1.177    | Mean : 0.6548   | Mean : 15.18   | Mean : 0.7626    | Mean : 15.47   |
| 3rd Qu.: | 1.000    | 3rd Qu.: 1.0800 | 3rd Qu.: 15.76 | 3rd Qu.: -0.0300 | 3rd Qu.: 17.02 |
| Max. :   | 36.000   | Max. : 57.5500  | Max. : 3328.12 | Max. : 951.8300  | Max. : 3589.41 |

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.

```

59 If you don't want to lose the other information from the incomplete observation, then you can fill in the missing values with the mean or
median value from that column. To do that, you can use the ifelse() function, which is identical to the =IF() function in Excel. We'll do
that within the mutate function, and then calculate the value for TotalDue.
60 - |{|
61 | ifelse
62 | j17L3 -- j17L3|}|
63 | mutate|
64 |   Tax = ifelse(is.na.Tax), mean(Tax, na.rm = T), Tax) # Could use median() instead of mean() if there are outliers
65 |   , TotalDue = NetTotal + Tax|
66 | }|
67 | summary(j17L3)
68 |
```

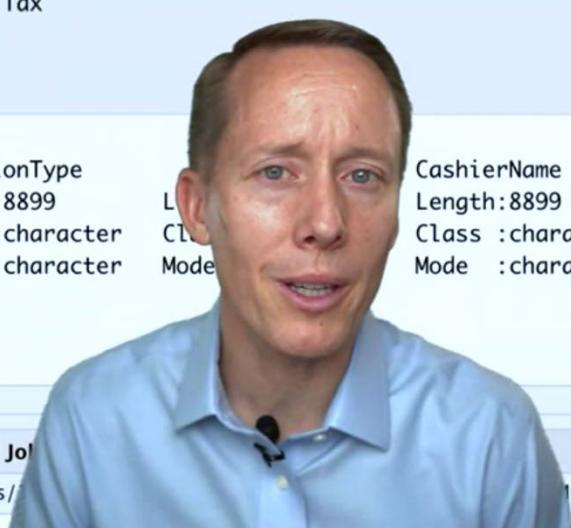
|          | Cost      | Price           |
|----------|-----------|-----------------|
| Min.     | -189.0400 | Min. : -322.810 |
| 1st Qu.: | 0.1100    | 1st Qu.: 4.500  |
| Median : | 0.1100    | Median : 12.020 |
| Mean :   | 0.2108    | Mean : 9.612    |
| 3rd Qu.: | 0.1100    | 3rd Qu.: 14.680 |
| Max. :   | 189.0300  | Max. : 24.630   |
| NA's     |           | : 562           |

|          | Tax     | TotalDue       |
|----------|---------|----------------|
| Min.     | -25.340 | Min. : -348.11 |
| 1st Qu.: | 0.460   | 1st Qu.: 5.25  |
| Median : | 1.040   | Median : 14.17 |
| Mean :   | 1.131   | Mean : 15.47   |
| 3rd Qu.: | 1.230   | 3rd Qu.: 17.02 |
| Max. :   | 261.260 | Max. : 3589.41 |

|          | Quantity | Modifiers       | Subtotal       | Discounts        | Net            |
|----------|----------|-----------------|----------------|------------------|----------------|
| Min. :   | 1.000    | Min. : -2.5200  | Min. : -322.80 | Min. : -0.0300   | Min. : -322.81 |
| 1st Qu.: | 1.000    | 1st Qu.: 0.0100 | 1st Qu.: 5.24  | 1st Qu.: -0.0300 | 1st Qu.: 4.50  |
| Median : | 1.000    | Median : 0.0100 | Median : 13.10 | Median : -0.0300 | Median : 12.02 |
| Mean :   | 1.177    | Mean : 0.6548   | Mean : 15.18   | Mean : 0.7626    | Mean : 15.47   |
| 3rd Qu.: | 1.000    | 3rd Qu.: 1.0800 | 3rd Qu.: 15.76 | 3rd Qu.: -0.0300 | 3rd Qu.: 17.02 |
| Max. :   | 36.000   | Max. : 57.5500  | Max. : 3328.12 | Max. : 951.8300  | Max. : 3589.41 |

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.



The image shows a video overlay of a man in a blue shirt speaking, positioned over a screenshot of an RStudio interface. The RStudio interface displays R code in the top pane and its output in the bottom pane. The code uses the ifelse function to handle missing values in a dataset named j17i\_3. The output shows the structure of the dataset, indicating character and mode for various columns like Time, OperationType, and CashierName.

```
60 ~ ``{r}
61 ?ifelse
62 j17i_3 <- j17i %>%
63   mutate(
64     Tax = ifelse(is.na(Tax), mean(Tax, na.rm = T), Tax) # Could use median()
65     , TotalDue = NetTotal + Tax
66   )
67 summary(j17i_3)
68 ~ ````
```

| Time             | OperationType    | CashierName      |
|------------------|------------------|------------------|
| Length:8899      | Length:8899      | Length:8899      |
| Class :character | Class :character | Class :character |
| Mode :character  | Mode :character  | Mode :character  |

64:17 Chunk 8

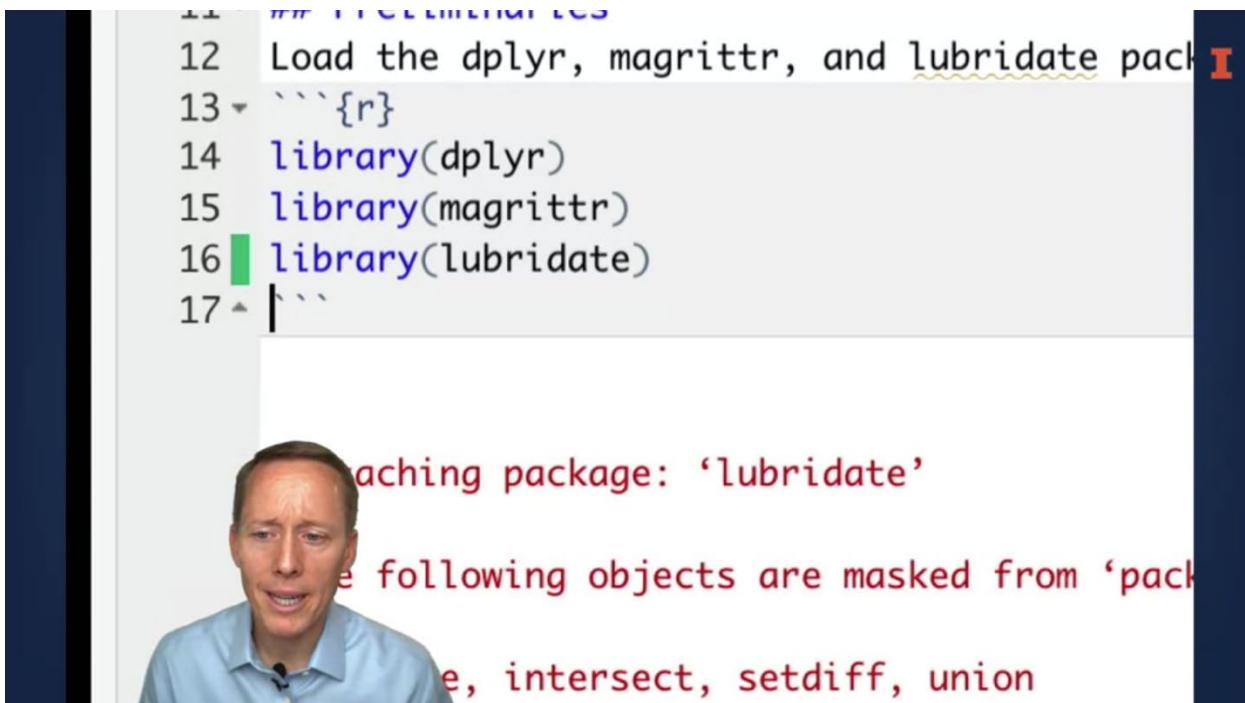
Console Terminal × R Markdown × Jobs

~/Box Sync/(Focus Area 4) Business Analytics/

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.



```
## FUTUREPACKAGES
12 Load the dplyr, magrittr, and lubridate pack I
13 ` ``{r}
14 library(dplyr)
15 library(magrittr)
16 library(lubridate)
17 ` ``
```

Attaching package: ‘lubridate’  
The following objects are masked from ‘package:base’:  
 date, intersect, setdiff, union

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

you can get errors. Notice that the resulting structure of `janm`, as well as the three newly computed columns. You can in January) and which dates are unexpectedly in the data

**M4\_11 Data Aggregation and S... 2.8 KB**

| Name                              | Size     | Modified      |
|-----------------------------------|----------|---------------|
| M4_06 NOTEBOOK Review and a...    | 2.4 KB   | Dec 17, 20... |
| .Rhistory                         | 17.2 KB  | Dec 2, 20...  |
| M4_05 Data Structure Wide Ver...  | 1.3 KB   | Dec 1, 20...  |
| M4_04 Data Structure Control V... | 2 KB     | Dec 1, 20...  |
| M4_14 Joining Data.nb.html        | 672.2 KB | Dec 1, 20...  |
| M4_14 Joining Data.Rmd            | 6.6 KB   | Dec 1, 20...  |
| M4_13 Stacking and Sorting Da...  | 667.8 KB | Nov 30, 20... |
| M4_13 Stacking and Sorting Da...  | 5.4 KB   | Nov 30, 20... |
| M4_12 Data Stacks.nb.html         | 655.2 KB | Nov 30, 20... |
| M4_12 Pivoting Dataframes Bet...  | 664.1 KB | Nov 30, 20... |
| M4_12 Pivoting Dataframes Bet...  | 3.8 KB   | Nov 30, 20... |

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.

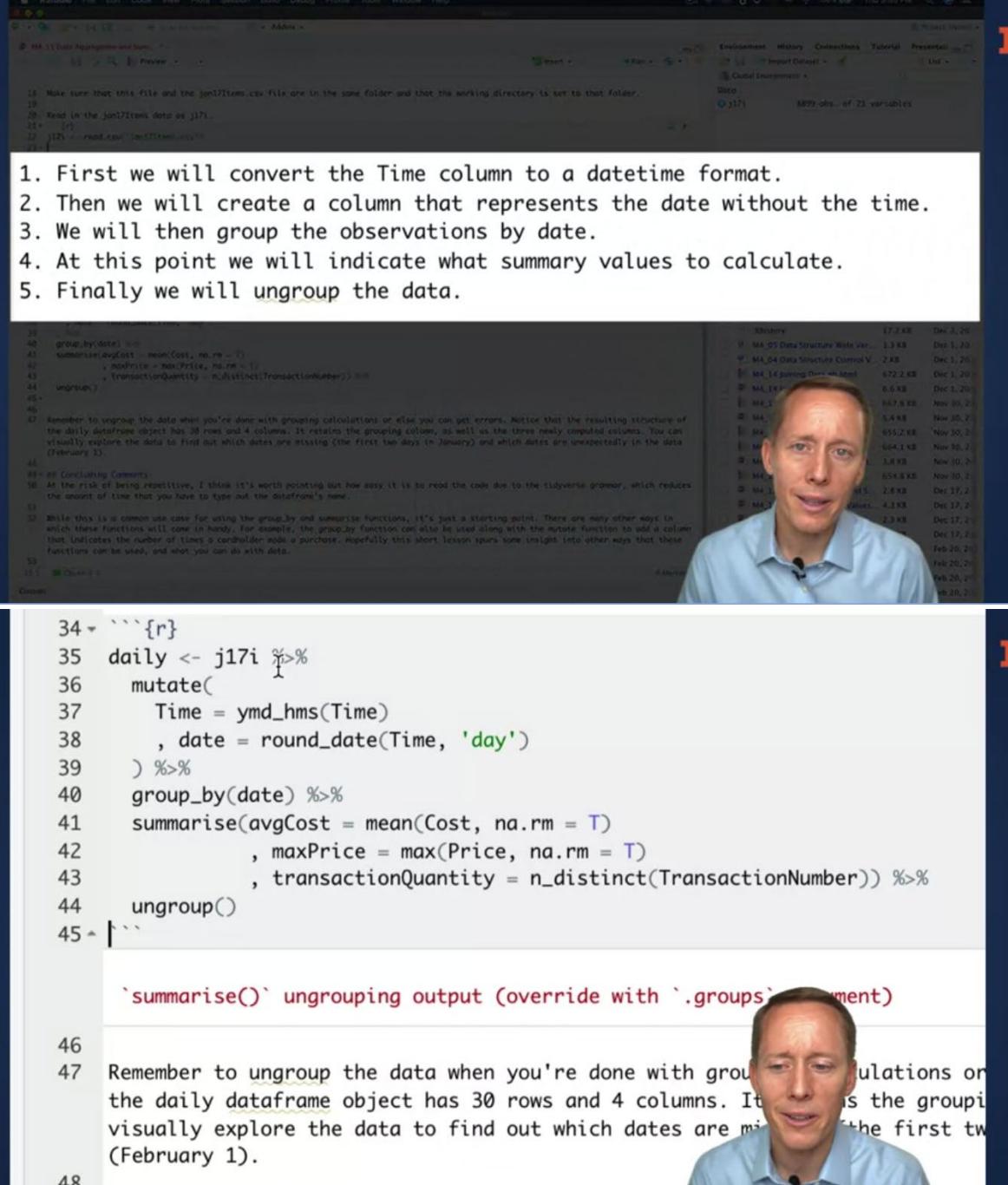
**Environment**   **History**   **Connections**   **Tutorial**   **Presentation**

**Import Dataset** | **Global Environment** | **List** | **C**

**Data**

j17i   8899 obs. of 21 variables

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.



MA 13 Data Aggregation and Summarise

18. Make sure that this file and the j17Items.csv file are in the same folder and that the working directory is set to that folder.

19.

20. Read in the j17Items.csv file as j17i.

21. `j17i <- read.csv('j17Items.csv')`

1. First we will convert the Time column to a datetime format.

2. Then we will create a column that represents the date without the time.

3. We will then group the observations by date.

4. At this point we will indicate what summary values to calculate.

5. Finally we will ungroup the data.

```
33
34 group_by(date) %>%
35 summarise(avgCost = mean(Cost, na.rm = T),
36           maxPrice = max(Price, na.rm = T),
37           transactionQuantity = n_distinct(TransactionNumber)) %>%
38 ungroup()
```

Remember to ungroup the data when you're done with grouping calculations or else you can get errors. Notice that the resulting structure of the daily\_dataframe object has 30 rows and 4 columns. It retains the grouping column, as well as the three newly computed columns. You can visually explore the data to find out which dates are missing (the first two days in January) and which dates are unexpectedly in the data (February 1).

## Concluding Comments

At the risk of being repetitive, I think it's worth pointing out how easy it is to read the code due to the tidyverse grammar, which reduces the amount of time that you have to type out the dataframe's name.

While this is a common use case for using the `group_by` and `summarise` functions, it's just a starting point. There are many other ways in which these functions will come in handy. For example, the `group_by` function can also be used along with the `mutate` function to add a column that indicates the number of times a cardholder made a purchase. Hopefully this short lesson spurs some insight into other ways that these functions can be used, and what you can do with data.

```
34 ````{r}
35 daily <- j17i %>%
36   mutate(
37     Time = ymd_hms(Time),
38     date = round_date(Time, 'day'))
39 ) %>%
40 group_by(date) %>%
41 summarise(avgCost = mean(Cost, na.rm = T),
42           maxPrice = max(Price, na.rm = T),
43           transactionQuantity = n_distinct(TransactionNumber)) %>%
44 ungroup()
45 ````
```

``summarise()` ungrouping output (override with ``.groups` argument)`

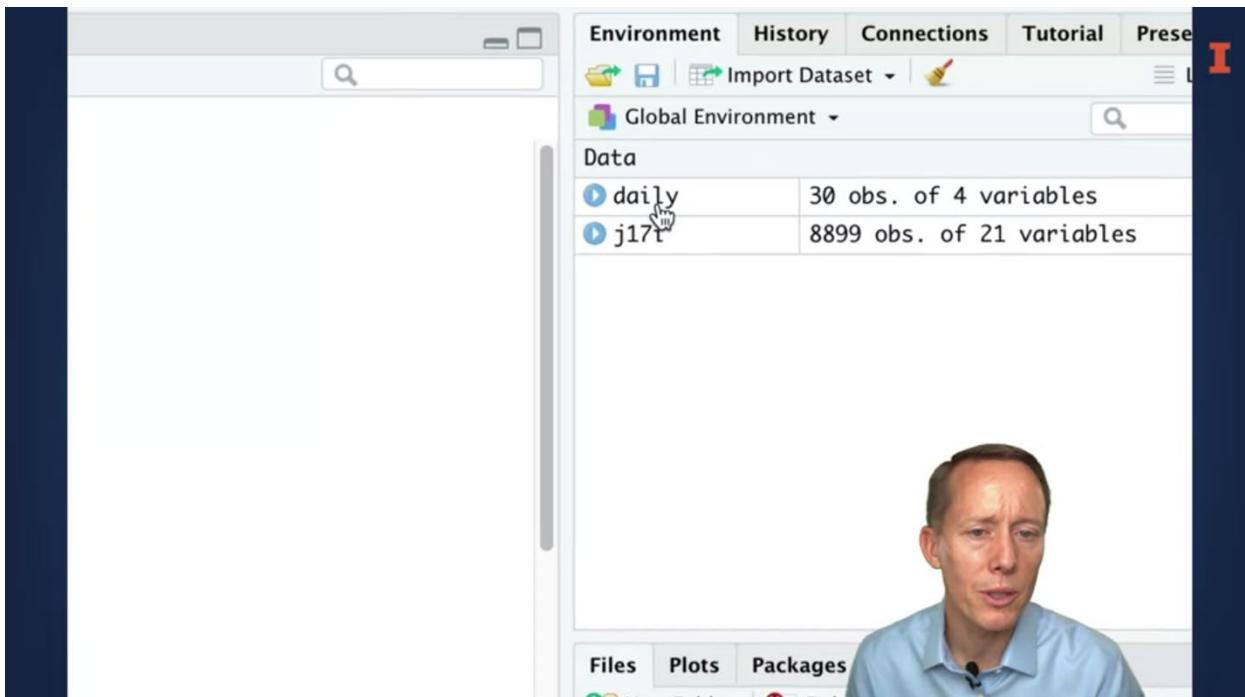
46

47 Remember to ungroup the data when you're done with grouping calculations or else you can get errors. Notice that the resulting structure of the daily\_dataframe object has 30 rows and 4 columns. It retains the grouping column, as well as the three newly computed columns. You can visually explore the data to find out which dates are missing (the first two days in January) and which dates are unexpectedly in the data (February 1).

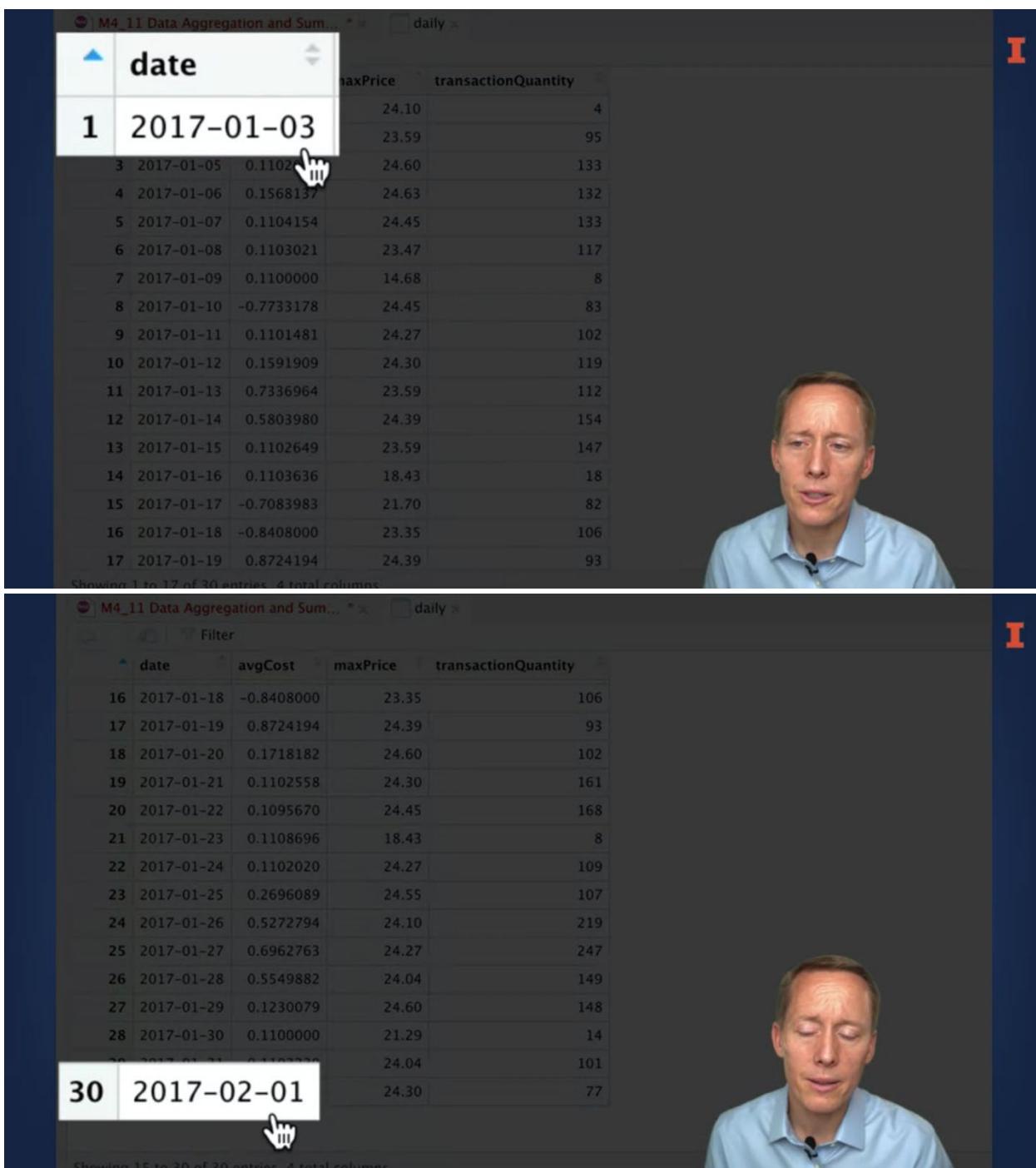
## Concluding Comments

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.



The image shows two data tables side-by-side, each with a header row and multiple data rows. The first table has columns: date, avgCost, maxPrice, and transactionQuantity. The second table has columns: date, avgCost, maxPrice, and transactionQuantity. Both tables show data from January 1st to February 1st, 2017. The first table ends at row 17 (2017-01-19), and the second table starts at row 16 (2017-01-18) and ends at row 30 (2017-02-01). A cursor icon is visible over the 'date' column header of the first table, and another cursor icon is visible over the 'date' column header of the second table.

|    | date       | avgCost    | maxPrice | transactionQuantity |
|----|------------|------------|----------|---------------------|
| 1  | 2017-01-03 |            | 24.10    | 4                   |
| 2  | 2017-01-05 | 0.1102137  | 23.59    | 95                  |
| 3  | 2017-01-06 | 0.1568137  | 24.60    | 133                 |
| 4  | 2017-01-07 | 0.1104154  | 24.63    | 132                 |
| 5  | 2017-01-08 | 0.1103021  | 24.45    | 133                 |
| 6  | 2017-01-09 | 0.1100000  | 23.47    | 117                 |
| 7  | 2017-01-10 | -0.7733178 | 14.68    | 8                   |
| 8  | 2017-01-11 | 0.1101481  | 24.45    | 83                  |
| 9  | 2017-01-12 | 0.1591909  | 24.27    | 102                 |
| 10 | 2017-01-13 | 0.7336964  | 23.59    | 119                 |
| 11 | 2017-01-14 | 0.5803980  | 24.39    | 154                 |
| 12 | 2017-01-15 | 0.1102649  | 23.59    | 147                 |
| 13 | 2017-01-16 | 0.1103636  | 18.43    | 18                  |
| 14 | 2017-01-17 | -0.7083983 | 21.70    | 82                  |
| 15 | 2017-01-18 | -0.8408000 | 23.35    | 106                 |
| 16 | 2017-01-19 | 0.8724194  | 24.39    | 93                  |

Showing 1 to 17 of 30 entries. 4 total columns.

|    | date       | avgCost    | maxPrice | transactionQuantity |
|----|------------|------------|----------|---------------------|
| 16 | 2017-01-18 | -0.8408000 | 23.35    | 106                 |
| 17 | 2017-01-19 | 0.8724194  | 24.39    | 93                  |
| 18 | 2017-01-20 | 0.1718182  | 24.60    | 102                 |
| 19 | 2017-01-21 | 0.1102558  | 24.30    | 161                 |
| 20 | 2017-01-22 | 0.1095670  | 24.45    | 168                 |
| 21 | 2017-01-23 | 0.1108696  | 18.43    | 8                   |
| 22 | 2017-01-24 | 0.1102020  | 24.27    | 109                 |
| 23 | 2017-01-25 | 0.2696089  | 24.55    | 107                 |
| 24 | 2017-01-26 | 0.5272794  | 24.10    | 219                 |
| 25 | 2017-01-27 | 0.6962763  | 24.27    | 247                 |
| 26 | 2017-01-28 | 0.5549882  | 24.04    | 149                 |
| 27 | 2017-01-29 | 0.1230079  | 24.60    | 148                 |
| 28 | 2017-01-30 | 0.1100000  | 21.29    | 14                  |
| 29 | 2017-01-31 | 0.1100000  | 24.04    | 101                 |
| 30 | 2017-02-01 |            | 24.30    | 77                  |

Showing 15 to 30 of 30 entries. 4 total columns.

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.



```
34 ````{r}
35 daily <- j17i %>%
36   mutate(
37     Time = ymd_hms(Time)
38     , date = round_date(Time, 'day')
39   ) %>%
40   group_by(date) %>%
41   summarise(avgCost = mean(Cost, na.rm = T)
42             , maxPrice = max(Price, na.rm = T)
43             , transactionQuantity = n_distinct(TransactionNumber)) %>%
44   ungroup()|
```

44:12 C Chunk 3

Console Terminal × R Markdown × Jobs ×

~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khandelwal)/Business Analytics Redesign/MOOC

ATTACHING package: lubridate

The following objects are masked from ‘package:base’:

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.

```
15 Load the dplyr, magrittr, and lubridate packa I
16 ````{r}
17 library(dplyr)
18 library(magrittr)
19 library(lubridate)
20 library(tidyr)
21 ````
```

Attaching package: 'tidyr'

The following object is masked from package



```
22 Make sure that this file and the jan17Items.co I
23
24 Read in the jan17Items data as j17i.
25 ````{r}
26 j17i <- read.csv('jan17Items.csv') I
27 ````

28
29 ``## Aggregate the Data at the Day Level
30 Aggregate the data into a single transaction
   transactions for each day.
31
32 ````{r}
33 daily <- j17i %>%
```



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.

The screenshot shows a data visualization interface with the following details:

- Global Environment** dropdown menu.
- Data** section showing a data frame named **j17i**.
- 8899 obs. of 21 variables** text indicating the size of the dataset.
- Time**, **OperationType**, **Barcode**, **CashierName**, **LineItem**, **Department**, **Category**, and **CardholderName** are the columns displayed in the preview.
- Filter** button at the top left of the preview table.
- Console** tab showing R code and output related to data processing.
- Terminal** tab showing command-line history.
- R Markdown** tab showing a link to a document.
- Jobs** tab showing a list of running processes.
- File** menu with options like **File**, **Plot**, **Print**, **Names**, **Value**, **...**, **Arguments**, **Data**, **cols**, and **meas**.

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.

```

32 ````{r}
33 daily <- j17i %>%
34   mutate(
35     Time = ymd_hms(Time)
36     , date = round_date(Time, 'day')
37   ) %>%
38   group_by(date) %>%
39   summarise(avgCost = mean(Cost, na.rm = T)
40             , maxPrice = max(Price, na.rm = T)
41             , transactionQuantity = n_distinct(TransactionNumber)) %>%
42   ungroup()
43 ````

`summarise()` ungrouping output (override with `.`groups` argument)

44
43:1  C Chunk 4

```

Console Terminal × R Markdown × Jobs ×

~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khandelwal) MOOC ATTACHING PACKAGE: ttruyr

M4\_12 Pivoting Dataframes Between... \* x j17i x daily x

Filter

|    | 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 | 0.04154    | 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                  |

Showing 1 to 17 of 30 entries, 4 total columns

Environment History Connections Tutorial Presentations

Data

- global environment
- daily 30 obs. of 4 variables
- j17i 8899 obs. of 21 variables

Files Photo Packages Help Viewer

R Pivot data between two data frames ...  
names\_to = "all",  
names\_pattern = NULL,  
names\_repair = list(),  
names\_transformer = list(),  
names\_repair = "check\_unique",  
values\_to = "value",  
values\_drop\_na = TRUE,  
values\_pivot = TRUE,  
values\_collapse = TRUE

Arguments

data cols

names\_to

by

using the name of the column to pivot the data from the data stored in the columns of data.

attaching or creating new data frames or environments

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.



## Pivot Longer

We're going to collapse the \*\*names\*\* for some columns into a single column, and the \*\*values\*\* from those columns into another single column. This will require duplicating the column(s) that identify the observation.

```
47  
48 <- r  
49 dailyLong <- daily %>%  
50 pivot_longer::cols = c(avgCost, maxPrice, transactionQuantity)  
51  
52 # Alternative approaches that don't work. See the tidy selects link in the  
53 dailyLong <- daily %>%  
54 pivot_longer::cols = avgCost:transactionQuantity  
55 dailyLong <- daily %>%  
56 pivot_longer::cols = c('avgCost', 'maxPrice', 'transactionQuantity')  
57  
58 Notice that the shape of this dataframe is 90 rows long and 3 columns  
59  
60 Let's specify what we want the names to be for the names_to and value  
61  
62 <- r  
63 dailyLong <- daily %>%  
64 pivot_longer::cols = avgCost:transactionQuantity  
65 , names_to = 'metrics'  
66 , values_to = 'vals'  
411 [2] # Chunks 4-2
```

Console Terminal R Markdown Jobs

~Box Sync/[Focus Area 4] Business Analytics/L. Course 1 (Guymon & Khandelwal)/MOOC\_P

The following object is masked from 'package:magrittr':

```
extract  
  
> j17i <- read.csv('joni7Items.csv')  
> View(j17i)  
> daily <- j17i %>%  
+ mutate(  
+   Time = ymd_hms(Time)  
+   , date = round_date(Time, 'day')  
+ ) %>%  
+ group_by(date) %>%
```

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.

The screenshot shows the RStudio interface. The Global Environment pane on the right lists three data frames:

| Object    | Description               |
|-----------|---------------------------|
| daily     | 30 obs. of 4 variables    |
| dailyLong | 90 obs. of 3 variables    |
| j17i      | 8899 obs. of 21 variables |

A cursor is hovering over the "dailyLong" entry. The top menu bar includes tabs for Environment, History, Connections, Tutorial, and Presentation, along with icons for file operations like Import Dataset, Save, and Print.

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

|    | date       | name                | value       |
|----|------------|---------------------|-------------|
| 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 | 133.0000000 |
| 10 | 2017-01-06 | avgCost             | 0.1568137   |
| 11 | 2017-01-06 | maxPrice            | 24.6300000  |

|    | date       | name                | value       |
|----|------------|---------------------|-------------|
| 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 | 133.0000000 |
| 10 | 2017-01-06 | avgCost             | 0.1568137   |
| 11 | 2017-01-06 | maxPrice            | 24.6300000  |
| 12 | 2017-01-06 | transactionQuantity | 132.0000000 |

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.

|           |                        |
|-----------|------------------------|
| daily     | 30 obs. of 4 variables |
| dailyLong | 90 obs. of 3 variables |

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.

```
+   , date = round_date(Time, 'day')
+ ) %>%
+ group_by(date) %>%
+ summarise(avgCost = mean(Cost, na.rm = T)
+           , maxPrice = max(Price, na.rm = T)
+           , transactionQuantity = n_distinct(Tr
+ ungroup()
`summarise()` ungrouping output (override with `.`gr
> View(daily)
> dailyLong <- daily %>%
+   pivot_longer(cols = c(avgCost, maxPrice, transa
> View(dailyLong)
> ?pivot_longer
```

Now, if you read the documentation for pivot longer.

The video player shows a man in a blue shirt speaking. To his right is a screenshot of an R documentation page titled "Argument type: tidy-select". The page includes a "Description" section and a "Overview of selection features" section. A callout box highlights the following points:

- : for selecting a range of consecutive variables.
- ! for taking the complement of a set of variables.
- & and | for selecting the intersection or the union of two sets of variables.
- c() for combining selections.

At the bottom of the callout box, there is a bullet point: • [everything\(.\)](#): Matches all variables.

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.

```

48 + ````{r}
49 dailyLong <- daily %>%
50   pivot_longer(cols = c(avgCost, maxPrice, transactionQuantity))
51
dailyLong <- daily %>%
  pivot_longer(cols = avgCost:maxPrice)
52
53 Notice that the shape of this dataframe is 90 rows long and 3 columns wide rather than 30 columns long and four rows wide.
54 Let's specify what we want the names to be for the names_to and values_to columns.
55
56 + ````{r}
57 dailyLong <- daily %>%
58   pivot_longer(cols = avgCost:transactionQuantity,
59                 , names_to = 'metrics',
60                 , values_to = 'vals')
61
62 54:25  Chunk 5

```

Console Terminal R Markdown Jobs

~/Box Sync/[Focus Area 4] Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to Use R in Business Analytics/rFiles/

```

> j17i <- read.csv('jan17Items.csv')
> View(j17i)
> daily <- j17i %>%
+   mutate(
+     Time = ymd_hms(Time)
+     , date = round_date(Time, 'day')
+   ) %>%
+   group_by(date) %>%

```

M4\_12 Pivoting Dataframes Between ... dailyLong daily

Filter

|    | avgCost    | maxPrice   | transactionQuantity |
|----|------------|------------|---------------------|
| 16 | 2017-01-18 | -0.8408000 | 23.35               |
| 17 | 2017-01-19 | 0.8724194  | 24.39               |
| 18 | 2017-01-20 | 0.1718182  | 24.60               |
| 19 | 2017-01-21 | 0.1102558  | 24.30               |
| 20 | 2017-01-22 | 0.1095670  | 24.45               |
| 21 | 2017-01-23 | 0.1108696  | 18.43               |
| 22 | 2017-01-24 | 0.1102020  | 24.27               |
| 23 | 2017-01-25 | 0.2696089  | 24.55               |
| 24 | 2017-01-26 | 0.5272794  | 24.10               |
| 25 | 2017-01-27 | 0.6962763  | 24.27               |
| 26 | 2017-01-28 | 0.5549882  | 24.04               |
| 27 | 2017-01-29 | 0.1230079  | 24.60               |
| 28 | 2017-01-30 | 0.1100000  | 21.29               |
| 29 | 2017-01-31 | 0.1103239  | 24.04               |
| 30 | 2017-02-01 | 0.1103478  | 24.30               |

Showing 15 to 30 of 30 entries, 4 total columns

Console Terminal R Markdown Jobs

~/Box Sync/[Focus Area 4] Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials

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.



```

44
45 ## Pivot Longer
46 Pivoting to a longer dataframe means that we're going to collapse the **names** for some columns into a single column
  from those columns into another single column. This will require duplicating the column(s) that identify the observation
47
48 ## {r}
49 dailyLong <- daily %>%
50   pivot_longer(cols = c(avgCost, maxPrice, transactionQuantity))
51
52 # Alternative approaches that also work. See the <tidy-select> link in the documentation
53 dailyLong <- daily %>%
54
55 dailyLong <- daily %>%
56   pivot_longer(cols = c('avgCost', 'maxPrice', 'transactionQuantity'))
57
58 Notice that the shape of this dataframe is 90 rows long and 3 columns wide rather than 30 columns long and four rows wide.
59 Let's specify what we want the names to be for the names_to and values_to columns.
60
61 ## {r}
62 dailyLong <- daily %>%
63   pivot_longer(cols = avgCost:transactionQuantity,
64                 , names_to = 'metrics',
65                 , values_to = 'vals')
66
67
68
69 
```

Console Terminal R Markdown Jobs

~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials



|    | date       | name                | value       |
|----|------------|---------------------|-------------|
| 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             | 0.1895842   |
| 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 | 133.0000000 |
| 10 | 2017-01-06 | avgCost             | 0.1568137   |
| 11 | 2017-01-06 | maxPrice            | 24.6300000  |
| 12 | 2017-01-06 | transactionQuantity | 132.0000000 |
| 13 | 2017-01-07 | avgCost             | 0.1104154   |
| 14 | 2017-01-07 | maxPrice            | 24.4500000  |
| 15 | 2017-01-07 | transactionQuantity | 133.0000000 |
| 16 | 2017-01-08 | avgCost             | 0.1103021   |
| 17 | 2017-01-08 | maxPrice            | 23.4700000  |

Showing 1 to 17 of 90 entries, 3 total columns

Console Terminal R Markdown Jobs

~/Box Sync/(Focus Area 4) Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials

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.

The R documentation for the 'pivot\_longer' function shows the 'names\_to' argument. It is described as a string specifying the name of the column to create from the data stored in the column names of 'data'. It can be a character vector, creating multiple columns if 'names\_sep' or 'names\_pattern' is provided. Two special values are mentioned: NA will discard that component of the name, and .value indicates that component of the name defines the name of the column containing the cell values, overriding 'values\_to'. Below this, there is a note about 'names\_sep' and 'names\_pattern'.

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.

```

61
62 #> `}`{r}
63 dailyLong <- daily %>%
64   pivot_longer(cols = avgCost:transactionQuantity,
65                 , names_to = 'metrics'
66                 , values_to = 'vals')
67 `}`{r}
68
69 ## Pivot Wider
70 Let's now pivot the long dataframe back to a wide version
    by running ?pivot_wider in the Console. You can see that
71
72 `}`{r}

```

64:50 C Chunk 6

Console Terminal × R Markdown × Jobs ×

~/Box Sync/(Focus Area 4) Business Analytics/1. Course

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.



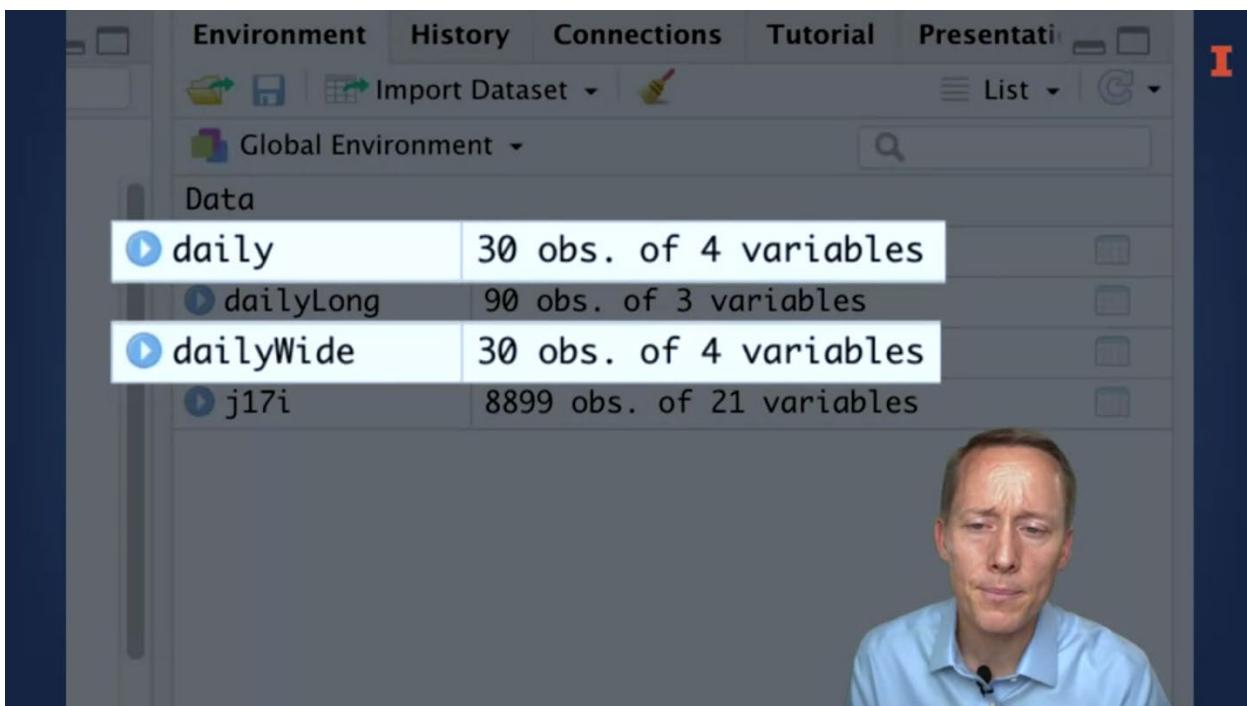
|   | date       | metrics             | vals        |
|---|------------|---------------------|-------------|
| 1 | 2017-01-03 | avgCost             | 0.1135294   |
| 2 | 2017-01-03 | maxPrice            | 24.1000000  |
| 3 | 2017-01-03 | transactionQuantity | 4.0000000   |
| 4 | 2017-01-04 | avgCost             | 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 | 133.0000000 |

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.

```
69 - ## Pivot Wider
70 Let's now pivot the long dataframe back to a wide version of it
    by running ?pivot_wider in the Console. You can see that it
71
72 - ````{r}
7 dailyWide <- dailyLong %>%
7   pivot_wider(names_from = metrics, values_from = vals)
75 -
76 A visual inspection confirms that the dailyWide dataframe is
    back to the wide format.
77
78
79 - ## Concluding Comments
80 It's nice that the pivot_wider and pivot_longer functions do
    what you can explore, but many of those options can be per-
81
82 Having a little bit of history using the tidyverse is al-
```



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?



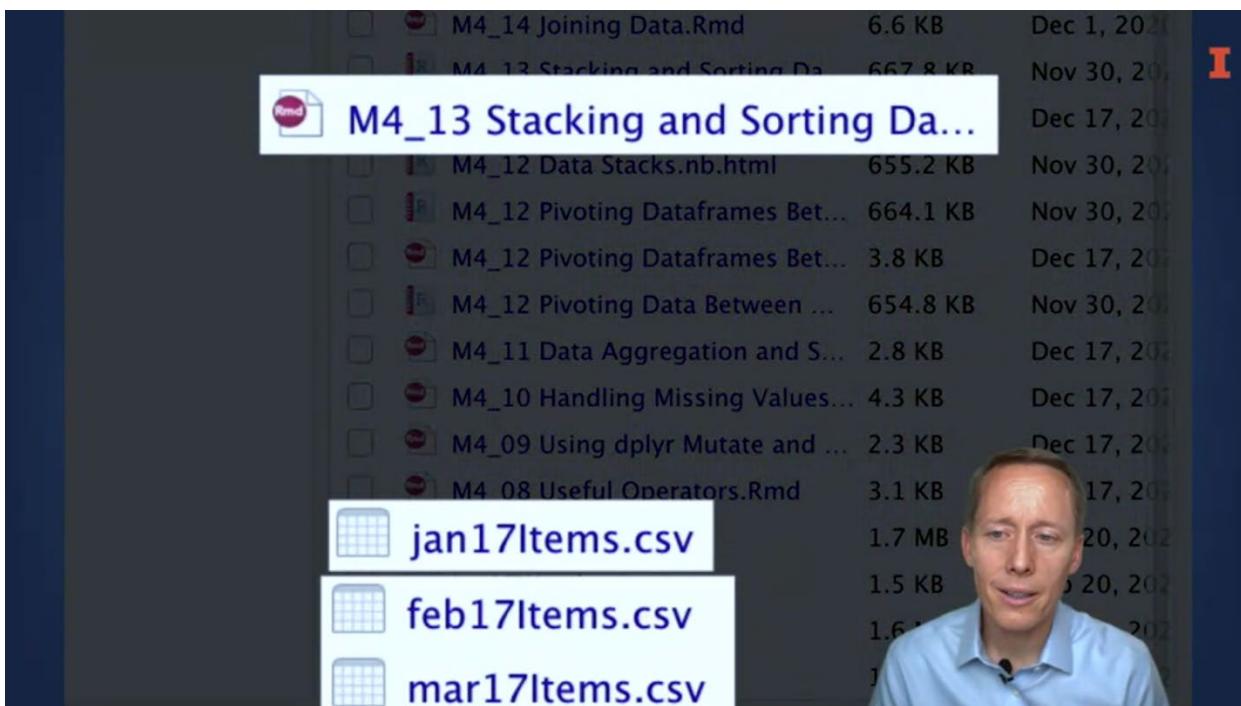
So we'll go ahead and run that, and we've got this new daily wide data frame object. You can see it's got the same dimensions as the daily data frame. And if we compare it to the daily data frame, you can see that they're exactly the same. Okay, so we just basically reverted back to that wide format 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.

```
13 The third function, arrange, is used for sorting data.  
14  
15 * ## Preliminaries  
16 Load the dplyr, and magrittr packages.  
17 * ` ``{r}  
18 library(dplyr)    |  
19 library(magrittr)  
20 * ````  
21 Make sure that this file and the jan17Items.csv file are in the same  
22  
23 Read in the jan17Items, feb17Items, and mar17Items.csv data.  
24 * ````{r}  
25 j17i <- read.csv('jan17Items.csv')  
26 f17i <- read.csv('feb17Items.csv')  
27 m17i <- read.csv('mar17Items.csv')  
28 * ````  
29  
30 * ## Stacking Rows of Data Using dplyr's bind_rows Function  
31 Now that we've read in the point-of-sale data for three consecutive months, let's review a couple of key points from the help documentation for bind_rows. Let's review a couple of key points from the help documentation for bind_rows.  
32  
33
```

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

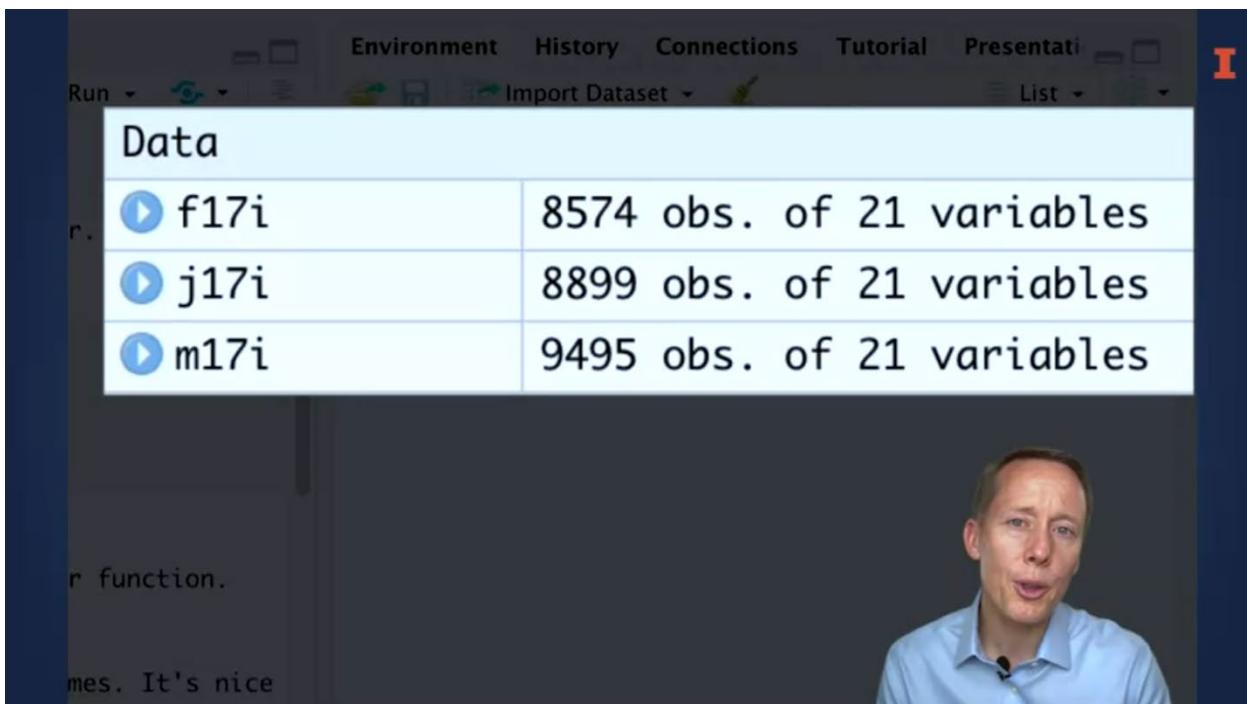


Then make sure that this our notebook file along with the jan 17 items dot CSV file, the feb 17 items dot CSV file and the mar 17 items dot CSV file are all in the same folder. And that the working directory is pointed to that folder.

```
18 library(dplyr)
19 library(magrittr)
20 ``
21 Make sure that this file and the jan17Items.csv file are in the same
22
23 Read in the jan17Items, feb17Items, and mar17Items.csv data.
24 ````{r}
25 j17i <- read.csv('jan17Items.csv') I
26 f17i <- read.csv('feb17Items.csv')
27 m17i <- read.csv('mar17Items.csv')
28 ``
29
30 ## Stacking Rows of Data Using dplyr's bind_rows Function
31 Now that we've read in the point-of-sale data for three consecutive months, let's review a couple of key points from the help documentation for bind_rows:
32
33 Notice that the first argument is ellipses, indicating that you can stack multiple dataframes at once. Let's
34
35 ````{r}
36 allItems <- bind_rows(j17i, f17i, m17i)
```



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

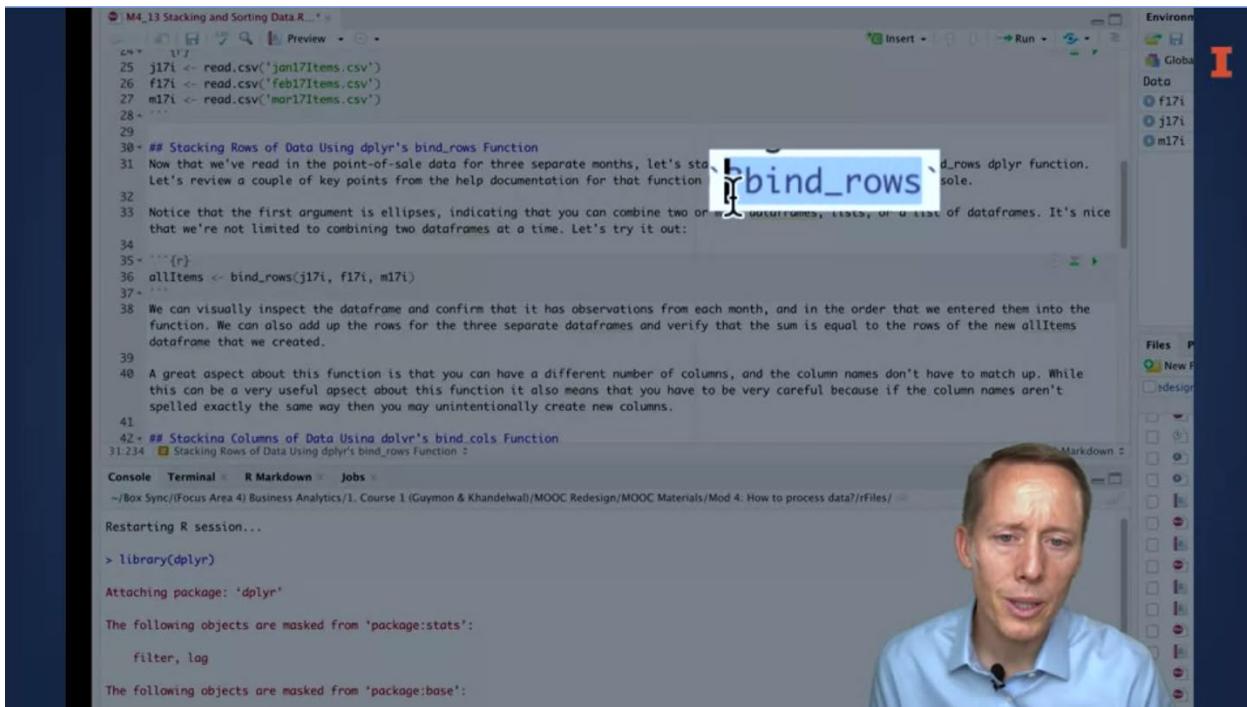


r function.

mes. It's nice

A video player window shows a man speaking.

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.



```

25 j17i <- read.csv('jan17Items.csv')
26 f17i <- read.csv('feb17Items.csv')
27 m17i <- read.csv('mar17Items.csv')
28 +
29
30 ## Stacking Rows of Data Using dplyr's bind_rows Function
31 Now that we've read in the point-of-sale data for three separate months, let's stack them together.
32 Let's review a couple of key points from the help documentation for that function
33 Notice that the first argument is ellipses, indicating that you can combine two or more dataframes, lists, or a list of dataframes. It's nice
34 that we're not limited to combining two dataframes at a time. Let's try it out:
35 +
36 allItems <- bind_rows(j17i, f17i, m17i)
37 +
38 We can visually inspect the dataframe and confirm that it has observations from each month, and in the order that we entered them into the
39 function. We can also add up the rows for the three separate dataframes and verify that the sum is equal to the rows of the new allItems
40 data frame that we created.
41
42 ## Stacking Columns of Data Using dplyr's bind_cols Function
43 Stacking Rows of Data Using dplyr's bind_rows Function :
```

Console Terminal R Markdown Jobs

Restarting R session...

> library(dplyr)

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

filter, lag

The following objects are masked from 'package:base':

A video player window shows a man speaking.

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

The screenshot shows the R documentation for the `bind_rows` function. The title is "Usage" and the code example is:

```
bind_rows(..., .id = NULL)
```

Below this is the "Arguments" section, which includes an ellipsis (...). A callout box highlights the word "Data frames to combine". The text explains that each argument can be a data frame, a list of data frames, or a list of data frames.

Further down, it says: "When row-binding, columns are matched by name, and any missing columns will be filled with NA." and "When column-binding, rows are matched by position, so all data frames must have the same number of rows. To match by value, not position, see [mutate-joins](#)".

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.

The screenshot shows the RStudio interface. In the console, the following code is run:

```
allItems <- bind_rows(j17i, f17i, m17i)
```

The "Environment" tab shows three data frames: j17i (8574 obs. of 21 variables), f17i (8599 obs. of 21 variables), and m17i (9495 obs. of 21 variables).

The "Help" tab is open, showing the documentation for `bind_rows`. It defines the arguments as "Data frames to combine" and describes how columns are matched by name or position.

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`.

The screenshot shows an RStudio interface. On the left, the R console displays R code for reading CSV files and combining them into a single data frame named 'allItems'. The code includes comments explaining the use of dplyr's bind\_rows and bind\_cols functions. The right pane shows a preview of the 'allItems' data frame with four rows of data:

|            | Data                       |
|------------|----------------------------|
| ▶ allItems | 26968 obs. of 21 variables |
| ▶ f17i     | 8574 obs. of 21 variables  |
| ▶ j17i     | 8899 obs. of 21 variables  |
| ▶ m17i     | 9495 obs. of 21 variables  |

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.

The screenshot shows an RStudio interface. The R console at the bottom shows the same code as the previous screenshot. The main area displays a preview of the 'allItems' data frame as a table. The columns include Time, OperationType, BarCode, CashierName, LineItem, Department, Category, and CardholderName. The table contains approximately 200 rows of data, with the first few rows visible:

| Time                     | OperationType | Barcode | CashierName    | LineItem                        | Department | Category                        | CardholderName  |
|--------------------------|---------------|---------|----------------|---------------------------------|------------|---------------------------------|-----------------|
| 4180 2017-02-11T14:10:00 | SALE          |         | Wallace Kuiper | Beef and Broccoli Stir Fry      | general    | general                         | FIONNUALA OBER  |
| 4181 2017-02-11T14:05:00 | SALE          |         | Wallace Kuiper | Beef and Squash Kabob           | Kabobs     | Beef                            | AVETT PICKRELL  |
| 4182 2017-02-11T14:05:00 | SALE          |         | Wallace Kuiper | Salmon and Wheat Bran Salad     | Salad      | general                         | AVETT PICKRELL  |
| 4183 2017-02-11T14:03:00 | SALE          |         | Wallace Kuiper | Chicken and Onion Kabob         | Kabobs     | Chicken                         | FAYLEEN OKEKE   |
| 4184 2017-02-11T14:03:00 | SALE          |         | Wallace Kuiper | Beef and Broccoli Stir Fry      | general    | general                         | FAYLEEN OKEKE   |
| 4185 2017-02-11T14:00:00 | SALE          |         | Wallace Kuiper | Soya and Basil Salad            | Salad      | general                         | DELLAMAE BOLD   |
| 4186 2017-02-11T13:58:00 | SALE          |         | Wallace Kuiper | Beef and Apple Burgers          | Entrees    | Beef and Apple Burgers          |                 |
| 4187 2017-02-11T13:58:00 | SALE          |         | Wallace Kuiper | Salmon and Wheat Bran Salad     | Entrees    | Salmon and Wheat Bran Salad     |                 |
| 4188 2017-02-11T13:56:00 | SALE          |         | Wallace Kuiper | Beef and Apple Burgers          | Entrees    | Beef and Apple Burgers          | KELTEN VALER    |
| 4189 2017-02-11T13:56:00 | SALE          |         | Wallace Kuiper | Salmon and Wheat Bran Salad     | Salad      | general                         | KELTEN VALER    |
| 4190 2017-02-11T13:56:00 | SALE          |         | Wallace Kuiper | Coconut and Beef Vindaloo       | general    | general                         | KELTEN VALER    |
| 4191 2017-02-11T13:56:00 | SALE          |         | Wallace Kuiper | Aubergine and Chickpea Vindaloo | Entrees    | Aubergine and Chickpea Vindaloo | KELTEN VALER    |
| 4192 2017-02-11T13:55:00 | SALE          |         | Wallace Kuiper | Beef and Squash Kabob           | Kabobs     | Beef                            | BRYLEAH BROWNE  |
| 4193 2017-02-11T13:55:00 | SALE          |         | Wallace Kuiper | Lamb and Veggie Kabob           | Kabobs     | Lamb                            | KRISTIANO LAUNE |
| 4194 2017-02-11T13:54:00 | SALE          |         | Wallace Kuiper | Soya and Basil Salad            | Salad      | general                         | KRISTIANO LAUNE |
| 4195 2017-02-11T13:54:00 |               |         |                |                                 |            |                                 |                 |

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.

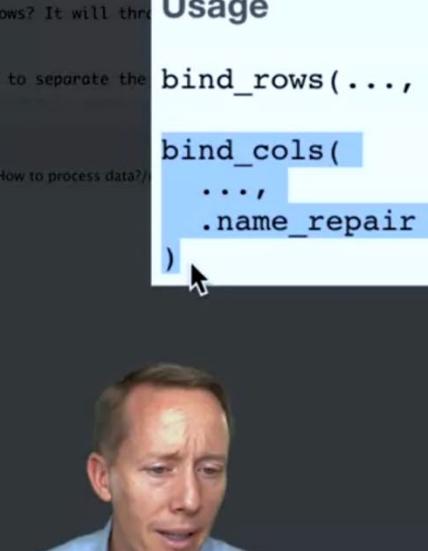
So a great aspect about this function is that if the column names are not the same or the same columns don't exist in each data frame, that's okay. I'll still stack them

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.

```
this can be a very useful aspect about this function it also means  
spelled exactly the same way then you may unintentionally create a problem  
41  
## Stacking Columns of Data Using dplyr's bind_cols Function  
43 The bind_cols function works similarly to the bind_rows function.  
     the same as for the bind_rows function.  
44  
45 To illustrate the bind_cols function, let's first separate the allItems data  
46 + ` ``{r}  
47 df1 <- allItems[,1:7]  
48 df2 <- allItems[,8:14]  
49 df3 <- allItems[,15:21]  
50 + `` ``{r}  
51  
52 Now, let's bind them back together, but in a different order.  
53 + `` ``{r}  
54 allItems2 <- bind_cols(df1, df3, df2)  
55 + `` ``{r}  
56 Comparing the number of rows and columns in the original data frame and the new data frame.
```



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.



This block contains a screenshot of a RStudio interface showing a help page for the `bind_rows` and `bind_cols` functions. The code examples are highlighted with blue boxes. A cursor arrow points to the closing parenthesis of the `bind_cols` example. The help page includes descriptions for the arguments `.name_repair`, `.id`, and `data.frames_to_combine`.

Having to separate the  
er of rows? It will thro  
ow.

having to separate the

Mod 4: How to process data?

## Usage

```
bind_rows(..., .id = NULL)
```

```
bind_cols(  
  ...,  
  .name_repair = c("unique", "universal", "check_  
)
```

... Data frames to combine:  
Each argument can either be a data frame, a list that could be a data frame, or a list of data frames.  
When row-binding, columns are matched by name, and any missing columns will be filled with NA.  
When column-binding, rows are matched by position, so all data frames must have the same number of rows. To match by value, not position, see [mutate\\_joins](#).

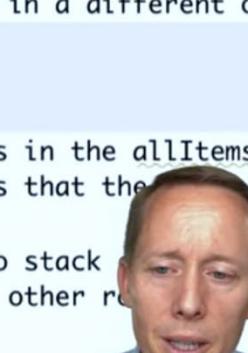
`.id` Data frame identifier.  
When `.id` is supplied, a new column of identifiers is created to link each row to its original data frame. The labels are taken

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.

```
46 ````{r}
47 df1 <- allItems[,1:7]
48 df2 <- allItems[,8:14]
49 df3 <- allItems[,15:21]
50 ````

51
52 Now, let's bind them back together, but in a different order.
53 ````{r}
54 allItems2 <- bind_cols(df1, df3, df2)
55 ````

56 Comparing the number of rows and columns in the allItems2 datafram
same shape. A visual inspection confirms that the shape of the co
57
58 What happens if you use the bind_cols to stack dataframes that have
row in which case it will fill in every other row with NA values
59
60 It's important to note that there are easier ways to stack them together again
```



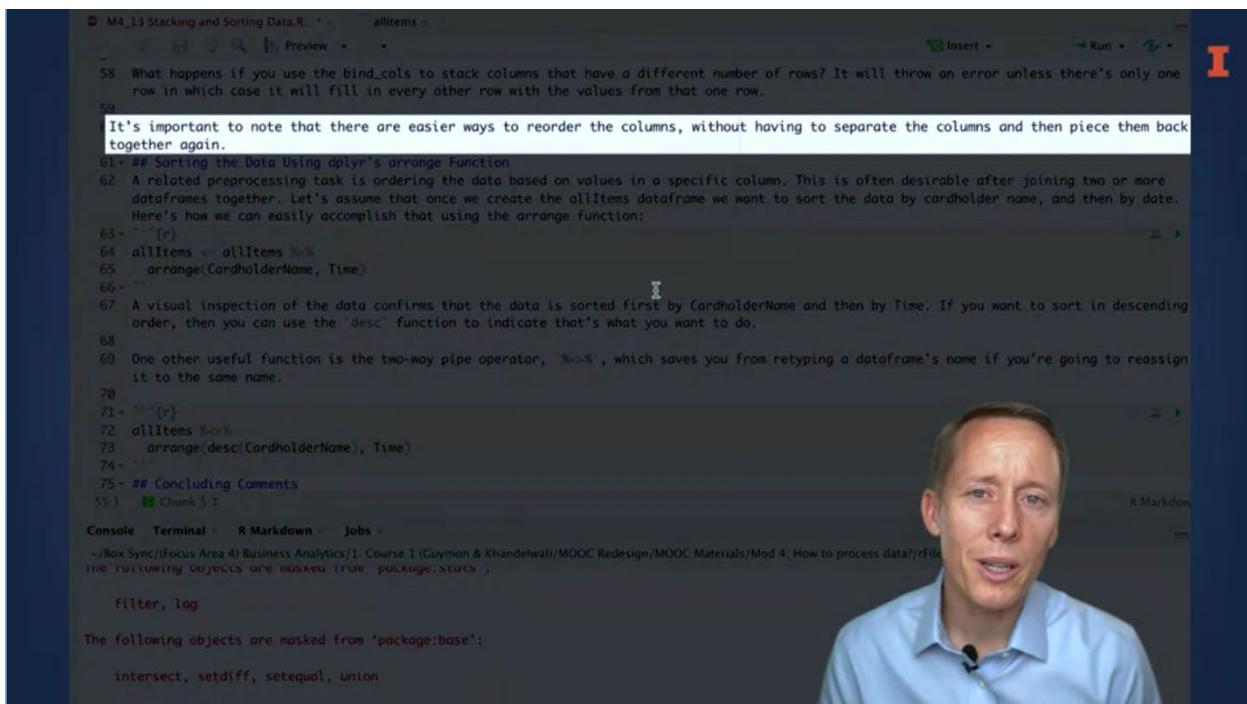
54:27 C Chunk 5 ↻

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.

|               | Data   |
|---------------|--|
| allItems      | 26968 obs. of 21 variables                                     |
| allItems2     | 26968 obs. of 21 variables                                     |
| Time          | chr "2017-01-26T21:18:00Z" "2017-01-26T20:...                  |
| OperationType | chr "SALE" "SALE" "SALE" "SALE" ...                            |
| BarCode       | chr "*" "*" "*" "*" ...  |
| CashierName   | chr "Nicholas Villines" "Carla Knot..."                        |
| LineItem      | chr "Glass Mug" "Lamb Chops" "Salmon a..."                     |
| Department    | chr "Beverage" "Entrees" "Entrees" ...                         |
| Category      | chr "Glass Bottle" "Lamb Chop" "Almo..."                       |
| Quantity      | int 1 1 1 1 1 1 1 1 1 1 ...                                    |
| Modifiers     | num 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 ...      |
| Subtotal      | num 3.74 13.39 13.91 2.89 4.39 13.39 13.91 2.89 4.39 13.39 ... |

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.



The image shows a video player window with a man in a light blue shirt speaking. In the background, a RStudio interface is visible. The code editor contains R code related to data manipulation. A callout box highlights a specific line of code: "It's important to note that there are easier ways to reorder the columns, without having to separate the columns and then piece them back together again."

```

58 What happens if you use the bind_cols to stack columns that have a different number of rows? It will throw an error unless there's only one
59 row in which case it will fill in every other row with the values from that one row.

59 It's important to note that there are easier ways to reorder the columns, without having to separate the columns and then piece them back
together again.

61 ## Sorting the Data Using dplyr's arrange Function
62 A related preprocessing task is ordering the data based on values in a specific column. This is often desirable after joining two or more
dataframes together. Let's assume that once we create the allItems dataframe we want to sort the data by cardholder name, and then by date.
Here's how we can easily accomplish that using the arrange function:
63 ~~~{r}
64 allItems <- allItems %>%
65   arrange(CardholderName, Time)
66 ~~~
67 A visual inspection of the data confirms that the data is sorted first by CardholderName and then by Time. If you want to sort in descending
order, then you can use the `desc` function to indicate that's what you want to do.
68
69 One other useful function is the two-way pipe operator, `%>%>%`, which saves you from retyping a dataframe's name if you're going to reassign
it to the same name.
70
71 ~~~{r}
72 allItems %>%>%
73   arrange(desc(CardholderName), Time)
74 ~~~
75 ## Concluding Comments
55:1  [4] Chunk 5 ±

```

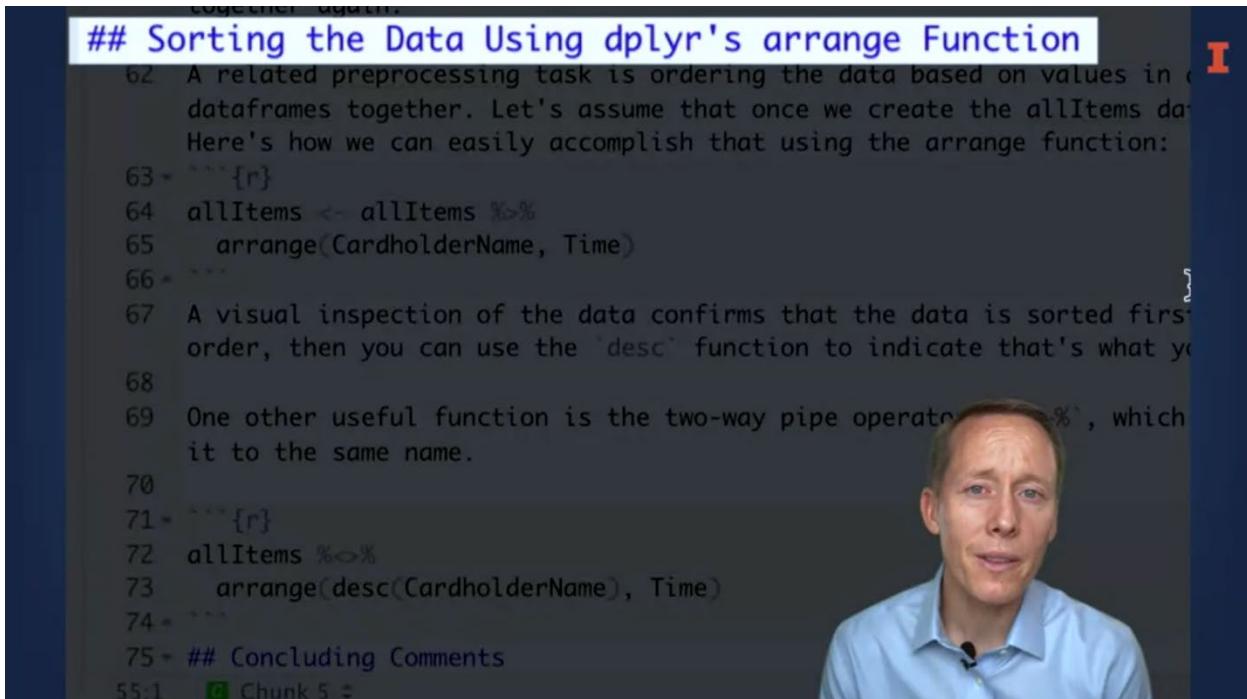
Console Terminal R Markdown Jobs

~/Box Sync/[Focus Area 4] Business Analytics/1\_Course 1 (Guymon & Khandelwal)/MOOC Redesign/MOOC Materials/Mod 4: How to process data/R/Files

The following objects are masked from 'package:base':

intersect, setdiff, setequal, union

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.



The image shows a video player window with a man in a light blue shirt speaking. In the background, a RStudio interface is visible. The code editor contains R code related to data manipulation. A callout box highlights the title of the section: "## Sorting the Data Using dplyr's arrange Function".

```

## Sorting the Data Using dplyr's arrange Function
62 A related preprocessing task is ordering the data based on values in a specific column. This is often desirable after joining two or more
dataframes together. Let's assume that once we create the allItems dataframe we want to sort the data by cardholder name, and then by date.
Here's how we can easily accomplish that using the arrange function:
63 ~~~{r}
64 allItems <- allItems %>%
65   arrange(CardholderName, Time)
66 ~~~
67 A visual inspection of the data confirms that the data is sorted first by CardholderName and then by Time. If you want to sort in descending
order, then you can use the `desc` function to indicate that's what you want to do.
68
69 One other useful function is the two-way pipe operator, `%>%>%`, which saves you from retyping a dataframe's name if you're going to reassign
it to the same name.
70
71 ~~~{r}
72 allItems %>%>%
73   arrange(desc(CardholderName), Time)
74 ~~~
75 ## Concluding Comments
55:1  [4] Chunk 5 ±

```

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.

```
63 - ``{r}
64 allItems <- allItems %>%
65   arrange(CardholderName, Time)
66 -
67 A visual inspection of the data confirms that
order, then you can use the `desc` function to
68
69 One other useful function is the two-way pipe
it to the same name.
70
71 - ``{r}
72 allItems %>>%
73   arrange(desc(CardholderName))
74 - ``-
```



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.

| OperationType | Barcode | CashierName    | LineItem                    | Department | Category       | CardholderName |
|---------------|---------|----------------|-----------------------------|------------|----------------|----------------|
| SALE          |         | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Nan                         | Sides      | Nan            | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Chutney                     | Sides      | Chutney        | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Yogurt                      | Sides      | Yogurt         | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Lamb and Veggie Kabobs      | Kabobs     | Lamb           | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Yogurt                      | Sides      | Yogurt         | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Rice                        | Sides      | Rice           | AADHI AUBRY    |
| SALE          |         | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Wallace Kasper | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Sharon Hill    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Sharon Hill    | Lamb Chops                  | Entrees    | Lamb Chops     | AADHI AUBRY    |
| SALE          |         | Sharon Hill    | Nan                         | Sides      | Nan            | AADHI AUBRY    |
| SALE          |         | Sharon Hill    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |
| SALE          |         | Karen Castro   | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY    |

```

Showing 2 to 23 of 26,000 entries. 22 total rows

Console Terminal R Markdown Jobs
--Box Sync/[Public Area]/Business Analytics/2. Course 1/Guymon & Khandelwal/MOOC Redesign/MOOC Materials/Mid 4

The following objects are masked from "package:base":
  intersect, setdiff, setequal, union

> library(magrittr)
> y17i <- read.csv("Jan17Items.csv")
> f17i <- read.csv("Feb17Items.csv")
> m17i <- read.csv("Mar17Items.csv")
> #Bind rows
> allItems <- bind_rows(y17i, f17i, m17i)
> allItems <- allItems[, -c(1, 2)]
> df1 <- allItems[1:14]
> df2 <- allItems[15:14]
> df3 <- allItems[15:21]
> allItems2 <- bind_cols(df1, df3, df2)
> allItems <- allItems %>%
>   arrange(CardholderName, Time)
>

```

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.

| Time                 | CashierName    | LineItem                    | Department | Category       | CardholderName      |
|----------------------|----------------|-----------------------------|------------|----------------|---------------------|
| 2017-03-17T19:16:00Z | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Nan                         | Sides      | Nan            | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Chutney                     | Sides      | Chutney        | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Yogurt                      | Sides      | Yogurt         | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Lamb and Veggie Kabobs      | Kabobs     | Lamb           | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Tzatziki                    | Sides      | Yogurt         | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Rice                        | Sides      | Rice           | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Rachael Price  | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHI AUBRY         |
| 2017-03-17T19:16:00Z | Wallace Kasper | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AADHVIKA DITCHFIELD |
| 2017-03-17T19:16:00Z | Sharon Hill    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AAGNA DRUM          |
| 2017-03-17T19:16:00Z | Sharon Hill    | Lamb Chops                  | Entrees    | Lamb Chops     | AAGNA DRUM          |
| 2017-03-17T19:16:00Z | Sharon Hill    | Nan                         | Sides      | Nan            | AAGNA DRUM          |
| 2017-03-17T19:16:00Z | Sharon Hill    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AAGNA DRUM          |
| 2017-03-17T19:16:00Z | Karen Castro   | Salmon and Wheat Bran Salad | Entrees    | Salmon and Whe | AAHLIYAH TRONGONE   |

```

> df1 <- allItems[1:14]
> df2 <- allItems[15:21]
> allItems2 <- bind_cols(df1, df3, df2)
> allItems <- allItems %>%
>   arrange(CardholderName, Time)
>

```

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.

```

library(dplyr)
# What happens if you use the bind_cols to stack columns that have a different number of rows? It will throw an error unless there's only one row in which case it will fill in every other row with the values from that one row.
# It's important to note that there are easier ways to reorder the columns, without having to separate the columns and then piece them back together again.
## Sorting the Data Using dplyr's arrange Function
# A related preprocessing task is ordering the data based on values in a specific column. This is often desirable after joining two or more dataframes together. Let's assume that once we create the allItems dataframe we want to sort the data by cardholder name, and then by date. Here's how we can easily accomplish that using the arrange function:
# A visual inspection of the data confirms that the data is sorted first by CardholderName and then by Time. If you want to sort in descending order, then you'd use the desc function.
# One other useful operator is `arrange_` which saves you from retyping a dataframes name if you're going to reassign it to the same name.
# An additional benefit of using the pipe operator is that it allows us to chain multiple operations together.
# Another benefit of using the pipe operator is that it allows us to chain multiple operations together.
# All Consulting Comments
# Saving the Data Using dplyr's arrange Function

```

The following objects are masked from 'package:base':

```

intersect, setdiff, setequal, union

> library(dplyr)
> j171 <- read.csv('jan17Items.csv')
> f171 <- read.csv('feb17Items.csv')
> m171 <- read.csv('mar17Items.csv')
> #bind_rows
> allItems <- bind_rows(j171, f171, m171)
> #view(allItems)
> df1 <- allItems[1:7]
> df2 <- allItems[8:14]
> df3 <- allItems[15:21]
> allItems2 <- bind_cols(df1, df2, df3)
> allItems <- allItems %>
>   arrange(CardholderName, Time)
>

```

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

```

library(dplyr)
# What happens if you use the bind_cols to stack columns that have a different number of rows? It will throw an error unless there's only one row in which case it will fill in every other row with the values from that one row.
# It's important to note that there are easier ways to reorder the columns, without having to separate the columns and then piece them back together again.
## Sorting the Data Using dplyr's arrange Function
# A related preprocessing task is ordering the data based on values in a specific column. This is often desirable after joining two or more dataframes together. Let's assume that once we create the allItems dataframe we want to sort the data by cardholder name, and then by date. Here's how we can easily accomplish that using the arrange function:
# A visual inspection of the data confirms that the data is sorted first by CardholderName and then by Time. If you want to sort in descending order, then you'd use the desc function.
# One other useful operator is `arrange_` which saves you from retyping a dataframes name if you're going to reassign it to the same name.
# An additional benefit of using the pipe operator is that it allows us to chain multiple operations together.
# Another benefit of using the pipe operator is that it allows us to chain multiple operations together.
# All Consulting Comments
# Saving the Data Using dplyr's arrange Function

```

The following objects are masked from 'package:base':

```

intersect, setdiff, setequal, union

> library(dplyr)
> j171 <- read.csv('jan17Items.csv')
> f171 <- read.csv('feb17Items.csv')
> m171 <- read.csv('mar17Items.csv')
> #bind_rows
> allItems <- bind_rows(j171, f171, m171)
> #view(allItems)
> df1 <- allItems[1:7]
> df2 <- allItems[8:14]
> df3 <- allItems[15:21]
> allItems2 <- bind_cols(df1, df2, df3)
> allItems <- allItems %>
>   arrange(CardholderName, Time)
>

```

the two-way pipe operator, `%<>%`,

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.

```
71 - ``{r}
72 allItems %<>%   I
73   arrange(desc(CardholderName), Time)
74 -
75 ## Concluding Comments
76 In conclusion, these functions to bind and so
    I use the bind_rows function because I often
    weather data. When this is the case, I use a
    another lesson, though.
77
```

73:23 C Chunk 7 ▾

Console Terminal × R Markdown × Join

~/Box Sync/(Focus Area 4) Business Analytics/



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.

| OperationType | Barcode | CardholderName    | LineItem                    | Department | Category               |
|---------------|---------|-------------------|-----------------------------|------------|------------------------|
| 00.00Z SALE   |         | Wallace Kuiper    | Beef and Broccoli Stir Fry  | general    | general                |
| 05.00Z SALE   |         | Cheryl Williams   | Salmon and Wheat Bran Salad | Entrees    | Salmon and Wheat Bran  |
| 05.00Z SALE   |         | Kathlene Starn    | Beef and Squash Kabob       | Kabobs     | Beef                   |
| 05.00Z SALE   |         | Kathlene Starn    | Beef and Squash Kabob       | Kabobs     | Beef                   |
| 43.00Z SALE   |         | Katherine Roth    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Wheat Bran  |
| 43.00Z SALE   |         | Katherine Roth    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Wheat Bran  |
| 43.00Z SALE   |         | Katherine Roth    | Lamb Chops                  | Entrees    | Lamb Chops             |
| 43.00Z SALE   |         | Katherine Roth    | Naan                        | Sides      | Naan                   |
| 43.00Z SALE   |         | Katherine Roth    | Yogurt                      | Sides      | Yogurt                 |
| 43.00Z SALE   |         | Katherine Roth    | Beef and Squash Kabob       | Kabobs     | Beef                   |
| 43.00Z SALE   |         | Katherine Roth    | Lamb Chops                  | Entrees    | Lamb Chops             |
| 43.00Z SALE   |         | Katherine Roth    | Salmon and Wheat Bran Salad | Entrees    | Salmon and Wheat Bran  |
| 43.00Z SALE   |         | Katherine Roth    | Naan                        | Sides      | Naan                   |
| 43.00Z SALE   |         | Katherine Roth    | Fountain Drink              | Beverage   | Fountain               |
| 12.00Z SALE   |         | Nicholas Williams | Chicken and Onion Kabob     | Kabobs     | Chicken                |
| 33.00Z SALE   |         | Rachael Price     | Beef and Apple Burgers      | Entrees    | Beef and Apple Burgers |

Showings 1 to 16 of 26,968 entries. 21 total columns

```

Console Terminal R Markdown Jobs
<File Sync>/Focus Area 4 Business Analytics/1. Course 1 (Guymon & Khandelwal)/MOOC Materials/MOD 4: How to Join Data Frames by Rows and Columns.Rmd
  1 running 0 remaining 0 suspended 0 failed 0 package index
  Intersect, setdiff, setequal, union
  > library(magrittr)
  > j171 <- read.csv('joni171Items.csv')
  > f171 <- read.csv('fobi171Items.csv')
  > m171 <- read.csv('mori171Items.csv')
  > #bind_rows
  > allItems <- bind_rows(j171, f171, m171)
  > View(allItems)
  > df1 <- allItems[,1:7]
  > df2 <- allItems[,8:14]
  > df3 <- allItems[,15:21]
  > allItems2 <- bind_cols(df1, df3, df2)
  > allItems <- allItems %>%
  >   arrange(CardholderName, Time)
  > allItems %>%
  >   arrange(desc(CardholderName), Time)
  >
  
```

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 lubridate packages.

```
26 Read in the jan17Items and jan17Weather data, and make
27 ````{r}
28 j17i <- read.csv('jan17Items.csv') %>%
29   mutate(
30     Time = ymd_hms(Time)
31   )
32 j17w <- read.csv('jan17Weather.csv', sep = '\t') %>%
33   mutate(
34     date = ymd_hms(date)
35   )
36 ````
```

37

38 **## Left Join on Aggregated Data**

39 For a join to work, we have to identify a common column to join on which will only be joined if they values in the primary key match. We can also ensure that the values in the primary key are unique.





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.

The screenshot shows a RStudio environment. On the left, a data frame titled 'Time' is displayed with 16 rows of data:

|    | Date                |
|----|---------------------|
| 1  | 2017-01-26 21:18:00 |
| 2  | 2017-01-26 20:30:00 |
| 3  | 2017-01-26 20:30:00 |
| 4  | 2017-01-26 20:30:00 |
| 5  | 2017-01-26 20:14:00 |
| 6  | 2017-01-26 20:14:00 |
| 7  | 2017-01-26 20:13:00 |
| 8  | 2017-01-26 20:02:00 |
| 9  | 2017-01-26 20:02:00 |
| 10 | 2017-01-26 20:00:00 |
| 11 | 2017-01-26 20:00:00 |
| 12 | 2017-01-26 20:00:00 |
| 13 | 2017-01-26 20:00:00 |
| 14 | 2017-01-26 19:52:00 |
| 15 | 2017-01-26 19:51:00 |
| 16 | 2017-01-26 19:49:00 |

The main workspace shows a data frame with columns: CashierName, LineItem, Department, Category, CardholderName, and Data. The 'Data' column contains two rows of JSON objects:

```

[{"id": 121, "text": "8899 obs. of 21 variables"}, {"id": 127, "text": "31 obs. of 5 variables"}]

```

The bottom right corner shows a file browser with several R Markdown files listed.

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.

The figure shows a screenshot of RStudio. On the left, a data frame titled "M4\_14 Joining Data.Rmd\*" is displayed with 17 rows and 5 columns: date, PRCP, SNOW, TMAX, and TMIN. The data includes dates from January 1 to 17, 2017, and corresponding weather values. On the right, the file browser shows several R files and a sub-directory "Materials" with various files like "M4\_14 Joining Data.html" and "M4\_14 Joining Data.Rmd".

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.

The video shows a man in a blue shirt speaking directly to the camera. In the background, a computer monitor displays an RStudio environment. The code in the console window is as follows:

```

Mr_14��售数据.R (27L) (27w)
11  mutate(
12  date = ymd_hms(date)
13
14 }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

For a join to work, we have to identify a column on which to match the data. This column is known as the primary key, and the values will only be joined if they values in the primary key column are identical. So one of the data wrangling tasks associated with joins is making sure that the values in the primary key column of both datasets are in the right format.
41 The obvious option for joining the weather data to the point-of-sale data is to use the Time and Date columns. Notice that while the Time
42 column in the j17i data and the date column in the j17w data are both the same data type, all of the date values are rounded to the beginning
43 hour, minute, and second of the day. There are probably few if any observations at midnight in our datasets.
44
45 Therefore, let's aggregate the point-of-sale data so that there's one observation for each day.
46
47 {
48   j17i %>% mutate(date = round(date, "day"))
49   daily = j17i %>
50   group_by(date) %>
51   summarise(
52     avgCost = mean(Cost, na.rm = TRUE),
53     maxPrice = max(Price, na.rm = TRUE),
54     transactionQuantity = n_distinct(TransactionNumber)
55   ) %>
56   ungroup()
57
58 # Notice that this data information has 889 observations and four variables... it does not contain dates for January 1 and January 2, but
59 # contains dates for January 3 through January 17.
60
61 # Check the data
62
63 # Summary of the data
64 library(dplyr)
65
66 Attaching package: 'dplyr'
67
68 The following objects are masked from 'package:base':
69
70   intersect, setdiff, union
71
72
73 j17i <- read.csv("j17iItems.csv") %>%
74   mutate(
75   date = ymd_hms(date)
76   )
77
78 j17w <- read.csv("j17iWeather.csv", sep = "\t") %>%
79   mutate(
80   date = ymd_hms(date)
81   )
82
83 View(j17i)
84 View(j17w)
85

```

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.

The screenshot shows the RStudio interface. At the top, there are three tabs: "M4\_14 Joining Data.Rmd" (active), "j17i", and "j17w". Below the tabs is a "Filter" button. A data frame table is displayed with columns: Time, OperationType, BarCode, CashierName, and LineItem. The data shows multiple transactions on January 26, 2017, involving various items like "Glass Mug", "Lamb Chops", and "Beef and Squash Kabob". In the bottom-left corner of the RStudio window, there is a small video player showing a man in a blue shirt speaking.

```

42
43 Therefore, let's aggregate the point-of-sale data so that there's one observation per day.
44 ``
45 j17i %>% mutate(date = round_date(Time, 'day'))
46 daily <- j17i %>%
47   group_by(date) %>%
48   summarise(avgCost = mean(Cost, na.rm = T),
49             , maxPrice = max(Price, na.rm = T)
50             , transactionQuantity = n_distinct(TransactionNumber)) %>%
51   ungroup()
52 ```
53 Notice that this daily dataframe has 30 observations and four variables. It does not include an observation for February. In contrast, the weather data has an observation for February.
54
36:1 [green] C Chunk 2

```

The "Console" tab at the bottom shows the following R session:

```

~/
> library(magrittr)
> library(lubridate)

Attaching package: 'lubridate'

```

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 create an aggregated data set for joining.

and the values will with joins is making that while the Time rounded to the beginning

```

M4_14 Joining Data.Rmd* daily j17i j17w
#> #> #> #>
#> #> #>
#> #> #>
```

|    | 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    | 141                 |
| 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.

```
include an observation for February. In contrast, the weather data has an obser
for February.

56 ````{r}
57 dailyLeft <- daily %>%
58   left_join(j17w, by = 'date')
59 ````

for the dates in the j17i data frame, which is the left one in this case because
the pipe symbol. Because there wasn't weather data for February 1, there are NA
62
63 ## Right Join on Aggregated Data
64 Now let's keep almost everything the same, except that we'll use the right_%
65 ````{r}
66 dailyRight <- daily %>%
67   right_join(j17w, by = 'date')
57:23 [C] Chunk 4 :~/
> j17L <- read.csv('juni7/items.csv') %>%
```

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 | 21.94 | 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 | 35.90 |       |
| 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 | 46.94 |       |
| 17 | 2017-01-19 | 0.8724194  | 24.39    |                     | 93   | 6.6141732 | 0.0000000 |       |       |

Showing 1 to 17 of 30 entries, 8 total columns

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.

Environment History Connections Tutorial Presentations

Import Dataset

Global Environment

Data

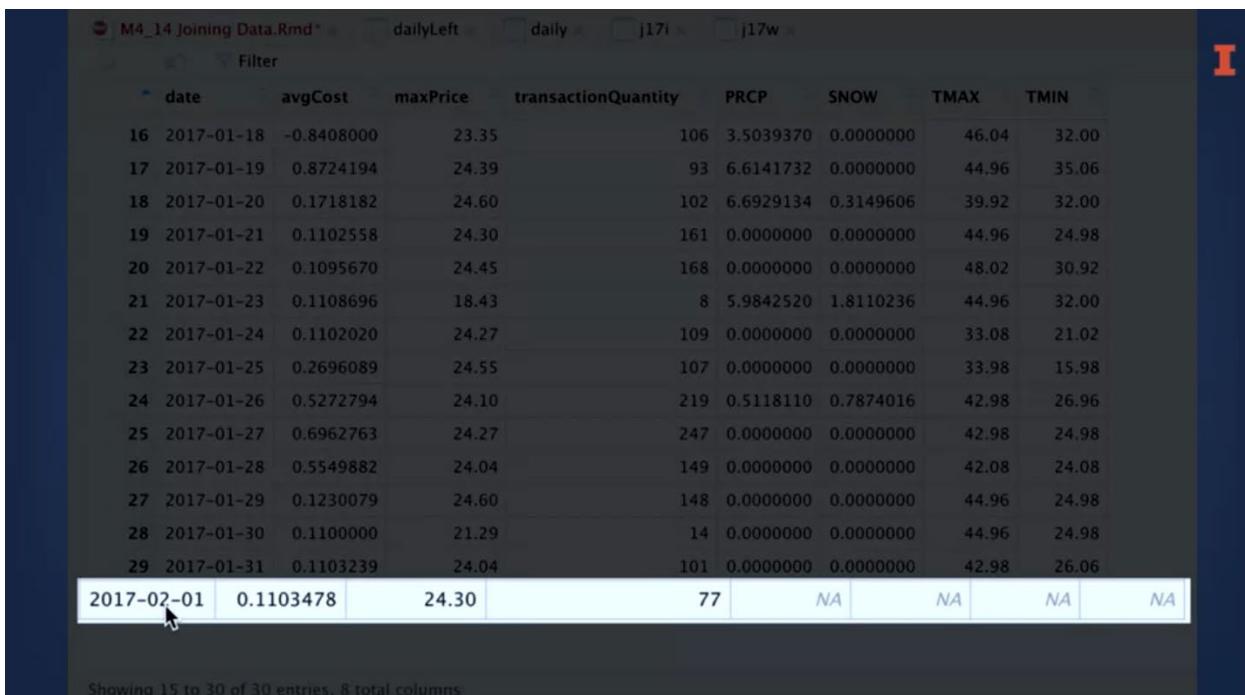
- daily 30 obs. of 4 variables
- dailyLeft 30 obs. of 8 variables
- j17i 8899 obs. of 22 variables
- j17w 31 obs. of 5 variables

Files Plots Packages Help Viewer

New Folder Delete Rename

edesign > MOOC Materials > Mod 4

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.



The screenshot shows an RStudio environment with a dark theme. A red 'I' logo is in the top right corner. The code editor window contains the following R code:

```
M4_14 Joining Data.Rmd * dailyLeft daily j17i j17w
Filter
#> #> #> #> #>
```

Below the code, a data frame named 'dailyLeft' is displayed with 30 rows of data. The columns are: date, avgCost, maxPrice, transactionQuantity, PRCP, SNOW, TMAX, TMIN. The last row of the data frame is highlighted.

|    | 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    |

Showing 15 to 30 of 30 entries. 8 total columns.

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 `NA`, 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.



```

55 Let's perform a left join with the point-of-sale data on the left and the weather data on the right
56 ````{r}
57 dailyLeft <- daily %>%
58   left_join(j17w, by = 'date')
59 ````

60
61 We now have a dailyLeft data frame with 30 observations and 8 variables. A visual inspection confirms
       for the dates in the j17i data frame, which is the left one in this case because that's what gets enclosed
       in the pipe symbol. Because there wasn't weather data for February 1, there are NA values for the weather
62

63 ````{r}
64 dailyRight <- daily %>%
65   right_join(j17w, by = 'date')
66 ````

67
68 ````

69
70
71 ## Inner Join on Aggregated Data
72 Let's compare the left and right join to an inner join.
73 ````{r}
74 dailyInner <- daily %>%
59:1  ↗ Chunk 4

```

Console Terminal Jobs ~/

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.



ght and see what we get:

irms that the weather values are there because they were entered into the left\_join function first, and not the right join function. So the right join function will return the same data frame, and no observations for data in columns, and there is not a value for February 1.

the right\_join function:

dailyRight 31 obs. of 8 variables  
j17w 31 obs.

Files Plots Packages Help Viewer

New Folder Delete Rename

edesign MOOC Materials Mod 4: H Name

M4\_14 Joining Data.r

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

|    | date       | avgCost   | maxPrice | transactionQuantity | PRCP      | SNOW      | TMAX  | TMIN  |
|----|------------|-----------|----------|---------------------|-----------|-----------|-------|-------|
| 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 | 42.98 | 24.08 |
| 27 | 2017-01-29 | 0.1230079 | 24.60    | 148                 | 0.0000000 | 0.0000000 | 42.98 | 24.98 |
| 28 | 2017-01-30 | 0.1100000 | 21.29    | 14                  | 0.0000000 | 0.0000000 | 42.98 | 24.98 |
| 29 | 2017-01-31 | 0.1103239 | 24.04    | 101                 | 0.0000000 | 0.0000000 | 42.98 | 24.06 |
| 30 | 2017-01-01 | NA        | NA       |                     | NA        | 0.7874016 | 34.08 | 21.02 |
| 31 | 2017-01-02 | NA        | NA       | ←                   | NA        | 0.1181102 | 32.00 | 24.98 |

Showing 16 to 31 of 31 entries, 8 total columns

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?

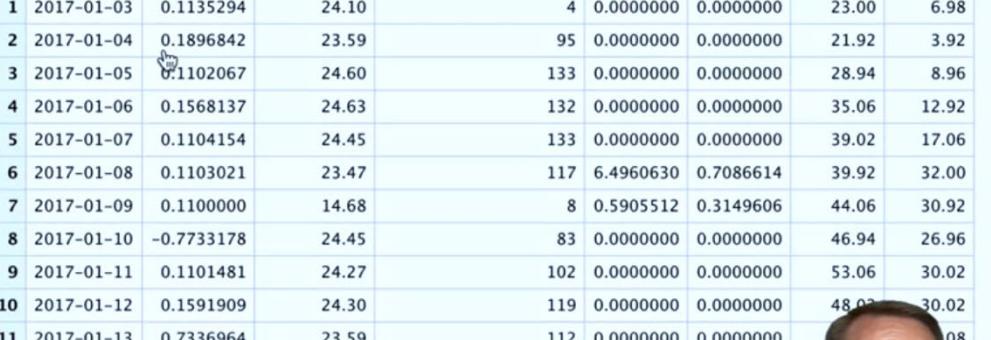


```
 63 ## Right Join on Aggregated Data
 64 Now let's keep almost everything the same, except that we'll replace the left_join function with the
 65 `right_join` function.
 66 dailyRight <- daily %>%
 67   right_join(j17w, by = 'date')
 68
 69 The dailyRight dataframe has 31 observations because there is weather data for every day in January
 70 and February. Notice that the rows for January 1 and January 2 have NA values for the point-of-sale column
 71.
 72
 73 ` ``{r}
 74 dailyInner <- daily %>%
 75   inner_join(j17w, by = 'date')
 76
 77
 78 ## Full Join on Aggregated Data
 79 Finally, let's evaluate a full join:
 80
 81 ` ``{r}
 82 dailyFull <- daily %>%
 83   full_join(j17w, by = 'date')
 84
 85
```

So that's a right join, 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.

The screenshot shows the RStudio interface. In the top-left corner, there's a tooltip for the variable `dailyInner`, which is described as having 29 observations. The tooltip also lists other variables: `dailyLeft` (30 obs., 9 variables), `dailyRight` (31 obs. of 8 variables), `j17i` (8899 obs. of 22 variables), and `j17w` (31 obs. of 5 variables). In the bottom right corner, a video frame of a man speaking is visible.

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 | 48.02 | 30.02 |
| 11 | 2017-01-13 | 0.7336964  | 23.59    |                     | 112  | 0.0000000 | 0.0000000 | 50.08 |       |
| 12 | 2017-01-14 | 0.5803980  | 24.39    |                     | 154  | 0.0000000 | 0.0000000 | 50.98 |       |
| 13 | 2017-01-15 | 0.1102649  | 23.59    |                     | 147  | 0.0000000 | 0.0000000 | 50.94 |       |
| 14 | 2017-01-16 | 0.1103636  | 18.43    |                     | 18   | 0.0000000 | 0.0000000 | 50.04 |       |
| 15 | 2017-01-17 | -0.7083983 | 21.70    |                     | 82   | 0.1181102 | 0.0000000 | 53.00 |       |
| 16 | 2017-01-18 | -0.8408000 | 23.35    |                     | 106  | 3.5039370 | 0.0000000 |       |       |
| 17 | 2017-01-19 | 0.8724194  | 24.39    |                     | 93   | 6.6141732 | 0.0000000 |       |       |

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.

```
70  
71+ ## Inner Join on Aggregated Data  
72 Let's compare the left and right join to an inner join.  
73+ ````{r}  
74 dailyInner <- daily %>%  
75   inner_join(j17w, by = 'date')  
76+ ````  
77 The dailyInner dataframe has 29 observations, which correspond to the 29 days in January that are in the data.  
78  
81+ ````{r}  
82 dailyFull <- daily %>%  
83   full_join(j17w, by = 'date')  
84+ ````  
NA values in the weather columns for February 1.  
86  
87+ ## Concluding Comments  
88 In conclusion, joins are really easy to do. As you use them more, you'll find yourself doing them all the time and other think carefully about what the shape of the combined dataframe should be. If you have many rows and many columns, it can be slow.  
83:8 [2] Chunk 7 :  
  
Console Terminal Jobs  
~/  
> j17i %>% mutate(date = round_date(time, "day"))  
> daily <- j17i %>%
```

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 |
| 19 | 2017-01-19 | 0.8724194  | 24.39    |                     | 93   | 6.6141732 | 0.0000000 | 44.96 |
| 20 | 2017-01-20 | 0.1718182  | 24.60    |                     | 102  | 6.6929134 | 0.3149606 | 39.92 |
| 21 | 2017-01-21 | 0.1102558  | 24.30    |                     | 161  | 0.0000000 | 0.0000000 | 44.96 |
| 22 | 2017-01-22 | 0.1095670  | 24.45    |                     | 168  | 0.0000000 | 0.0000000 | 48.02 |
| 23 | 2017-01-23 | 0.1108696  | 18.43    |                     | 8    | 5.9842520 | 1.8110236 | 44.96 |
| 24 | 2017-01-24 | 0.1102020  | 24.27    |                     | 109  | 0.0000000 | 0.0000000 | 33.08 |
| 25 | 2017-01-25 | 0.2696089  | 24.55    |                     | 107  | 0.0000000 | 0.0000000 | 33.98 |
| 26 | 2017-01-26 | 0.5272794  | 24.10    |                     | 219  | 0.5118110 | 0.7874016 | 42.98 |
| 27 | 2017-01-27 | 0.6962763  | 24.27    |                     | 247  | 0.0000000 | 0.0000000 | 42.98 |
| 28 | 2017-01-28 | 0.5549882  | 24.04    |                     | 149  | 0.0000000 | 0.0000000 | 42.08 |
| 29 | 2017-01-29 | 0.1230079  | 24.60    |                     | 148  | 0.0000000 | 0.0000000 | 44.96 |
| 30 | 2017-01-30 | 0.1100000  | 21.29    |                     | 14   | 0.0000000 | 0.0000000 | 44.96 |
| 31 | 2017-01-31 | 0.1103239  | 24.04    |                     | 101  | 0.0000000 | 0.0000000 | 42.98 |
| 32 | 2017-02-01 | 0.1103478  | 24.30    |                     | 77   | 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.

## Conceptual Lessons for Preprocessing Data

Frame a question so that it leads to actionable insight.

Aggregate data at the appropriate level.



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.

## Level of Aggregation

- The way the question is framed
- The manager's level of responsibility
- The level of granularity of the data
- The feasibility of implementing insights



Another factor that should influence the level at which the data is aggregated is the manager's 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.

**Conceptual Lessons  
for Preprocessing Data**

- Frame a question so that it leads to actionable insight.
- Aggregate data at the appropriate level.
- Shape the dataframe according to the analytic tool with which it will be processed.

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



**I**

## tidyverse

magrittr functions

- %>%
- %<>%



**I**

## tidyverse

tidyr functions

- pivot\_longer
- pivot\_wider



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/>