Representing, Processing, and Preparing Data

by Janani Ravi

Course Overview

Course Overview

Hi, my name is Janani Ravi, and welcome to this course on representing, processing, and preparing data. A little about myself. I have a master's degree in electrical engineering from Stanford and have worked at companies such as Microsoft, Google, and Flipkart. At Google, I was one of the first engineers working on real-time collaborative editing in Google Docs, and I hold four patents for its underlying technologies. I currently work on my own startup, Loonycorn, a studio for high quality video content. Data science and data modeling are fast emerging as crucial capabilities that every enterprise and every technologist must possess these days. As the process of actually constructing models becomes democratized, the _____ is shifting to using the right data and using the data right. In this course, you will gain the ability to correctly represent information from your domain as numeric data and get it into a form where the full capabilities of models can be leveraged. First, you'll learn how outliers and missing data can be dealt with in a perfectly sound manner. Next, you'll discover how to use spreadsheets, programming languages, and relational databases to work with your data. You'll see the different types of data that you may deal with in the real world, and how you can collect and integrate data to a common destination to eliminate silos. Finally, you'll round out the course by working with visualization tools that allow every member of an enterprise to work with data and extract meaningful insights.

When you're finished with this course, you will have the skills and knowledge to use the right data sources, cope with the data quality issues, and choose the right technologies to extract insights from your enterprise data.

Understanding Data Cleaning and Preparation Techniques

Module Overview

Hi, and welcome to this course on representing, processing, and preparing data. In this module, we'll understand data cleaning and preparation techniques that allows us to deal with problems that hinder analytics. In order to make data driven decisions, you first need to collect the data carefully. Once you've found the dots, you can connect them using analytics. This isn't always straightforward. There are various problems, such as bad data, missing data, duplicates, and the presence of outliers that hinder analytics. We'll then discuss common technology tools that we can use to work with data, whether it be spreadsheets for prototyping, Python programming for data analysis, or SQL to work with a database or a data warehouse. We'll then move on to discussing how we can deal with missing values in our dataset. We can either delete the record entirely or use imputation techniques. Our data might also compromise of outliers. These are values that don't fit in with the rest of the data. We'll see various techniques that we can use to cope with these outliers.

Prerequisites and Course Outline

Before we move on to the actual course contents and hands on demos, let's take a look at some of the prereqs that you need to have in order to make the most of your learning. This course assumes that you're familiar with Python programming, just the basics, and you know how to use data science libraries, such as NumPy and Pandas, and visualization libraries, such as Seaborn and Matplotlib. This course also assumes that you know how to work with Excel spreadsheets, you know how to use formulas in Excel, you know how to reference cells and ranges. This course will also discuss data preparation and analysis using SQL, other structure query language. We'll work

with data stored in relational databases using SQL. And finally, the only math and stats that you need to for this course is high school math. Let's take a look at how the course outline is structured. We'll first discuss data cleaning and preparation techniques. We'll see how to deal with missing data and outliers. We'll then move on to processing data using Excel spreadsheets and the Python programming language. We'll then see how we can collect data into a relational database to extract insights. We'll then process data using the structured query language. We'll work within Azure SQL database, set up on the Azure cloud. And finally, we'll see how we can represent insights gleaned from data using Microsoft Power BI and the Azure Data Studio.

Finding and Connecting Dots: Data Collection and Analysis

A powerful person once said, my mind is made up. Don't confuse me with facts. Well, that definitely makes the life of a data scientist easier. If you're a business or data analyst or you work with numbers in general, you'll often be called upon to present your thoughtful, fact-based point of view on important decisions, and this is where you really need to understand and work with data. Let's parse what exactly it means to have a thoughtful, fact-based point of view. A fact-based point of view essentially means that you have data to backup what you clean. Fact-based means built with painstakingly collected data. As a data-driven decision maker, it's not just enough to present facts. Your claims, your decisions, and your opinions have to be thoughtful. It has to be balanced and it has to weigh both the pros and cons of the path or course of action that you recommend. And finally, as someone who works with data, you'll often be asked for your point of view, that is your prediction, your recommendation, or your call to action. In order to make data-driven decisions, you really need to understand your data well, and there are two sets of statistical tools that you can use for this. Descriptive statistics, which are useful in helping your identify important elements in your dataset. Descriptive statistics are what you'd use to quantify and summarize the data that you have to work with. This data is typically a sample of the entire population. Descriptive statistics alone don't give you the full picture. They have to be used in combination with inferential statistics. These help explain the important elements that you've identified in your data, via relationships with other elements. Inferential statistical analysis infers properties of a population as a whole by testing just a sample. And these two broad categories explain the two hats that a data professional wears. One, they are responsible for finding the dots, identifying the important elements in a dataset, and another where you're responsible for connecting the dots, explaining those elements via relationships with other elements. When you're working in the real world, neither of these problems is particularly simple to solve. There are common problems that hinder analytics in both of these categories. Let's talk about finding

the dots first. Today, in this day and age, data is more and more plentiful. Often when you're seeking data to perform analysis, we might encounter a problem of plenty. However, data may not always be in the form that you want it to be. Careful handling might be needed before the data can be useful. You might find that in the real world, the data that you get to work with is often incomplete. It might have missing values, fields, or entire records that are missing. Or it might have outliers within it. Now some of these outliers might be genuine outliers. Genuine outliers are data points that are unusual, maybe they have too high or too low of value, but they are real data and contain information. Very often though, outliers might just be data points that have not been measured or calibrated correctly. Based on whether outliers are genuine or errors, you might have to deal with outliers differently. When you're collecting data for analytics, these are some of the common problems that you encounter. In addition, your data might contain duplicates or your data might just be bad, useful for the information that you're seeking. Let's say you've successfully managed to solve the finding the dots problem, that's when you'll move on to the next step, analysis or connecting the dots. Now there are a variety of tools that you could use for this. The easiest and the most intuitive tend to be spreadsheets, which represent your data in a tabular format. Or if you're familiar with programming, you might choose to work with data using a higher level programming language, such as Python or Java. Programming languages can be used to work with data beyond the prototyping keys. They tend to be more powerful and more robust. Programming languages give you the choice of in-memory processing of your data or distributed processing on a cluster of machines. Or if programming is not your cup of tea, you might want to analyze data using SQL, or the structured query language. SQL can be used with relational databases for small datasets or data warehouses, which hold very large data. The choice of technology that you need to process your data depends on what it is that you're trying to achieve. Let's say you're looking for very fast prototyping and an intuitive user interface, you'll choose to go with Microsoft Excel. Data analysis using Excel can be brittle though, it's not really great for production use. If you're looking for something more robust, you might choose to program in Python using Pandas and other data science libraries. Python is an easy to learn programming language, that's why it's so popular. It also has a variety of amazing data science libraries that you can harness. Python gives you fast prototyping in an interactive REPL environment where REPL stands for read, evaluate, print, loop. However, when you work with Python and data science libraries, this still gives you just in-memory processing of data, which isn't really great if your dataset is large. If you're unfamiliar with or uninterested in coding, you can work with data using the structured query language, or SQL. The Microsoft Azure platforms offers you SQL databases on the cloud. This is a managed service, making it very easy for you to work on relational data. This is not yet big data though, relational databases still work on

relatively small data sizes and you'll have the problem of data silos. There data may not be well integrated with other things in your organization. If you're working on large datasets and you're looking for fast prototyping, you can use the Python programming language with a framework such as Spark. Spark is part of the Hadoop ecosystem, it gives you distributed processing on a cluster of machines, but you still need to write code. A no code option that you can choose to work with big data is to use SQL with a data warehouse. A data warehouse integrates data from a variety of disparate sources. It can work with streaming data, and it also offers ML, or machine learning, integrations.

Dealing with Missing Values: Deletion and Imputation

We discussed earlier that a common problem that you encounter when you work with data in the real world is missing data. Now there's no way around this, so we have to deal with this. How? Well there are two broad ways to deal with missing data. One is where you throw up your hands and you say, I have no idea what to do with this record which has missing fields. You go ahead and delete that record. However, deleting data with missing fields and values can cause significant problems with analysis. So another technique that you can use to deal with missing data is imputation. In statistics, imputation is the process of replacing missing data with substituted values. Let's talk about using deletion to deal with missing data first. Deletion is also known as listwise deletion where you delete an entire record (row) if a single value or a field within that record has a missing value. The idea behind listwise deletion is simple and it's also simple to implement, but it can lead to bias. Simply deleting records can make the handling and analysis of the data more arduous, time consuming, and less efficient in general. But you'll often find that this is by far the most common way to deal with missing data because it's the simplest. There are several problems to using deletion to deal with missing data, the first of these is the fact that simply deleting records can reduce the sample size that we have to work with. For very large datasets, this may not be a problem, but if our dataset is small to begin with, if you delete records with missing data, you may not have that much data to analyze. The second is a problem of bias. If values are not missing at random across your entire dataset, if you get rid of missing data, this might introduce a significant bias in your analysis. It's quite possible that after you get rid of all records that have missing data, the dataset that you have to work with is not representative of the population as a whole, and that's not good. To overcome these drawbacks introduced due to data deletion, imputation is often used to deal with missing data. This is where you'll fill in missing column values, rather than deleting records with missing values. Data imputation refers to a category of techniques, it's not a single technique. Imputation methods can

be very simple indeed or can be arbitrarily complex. The simplest way to fill in missing values is to simply calculate the average of your column of data and fill in all missing fields with this average. Another technique is to use interpolation. You simply consider the values around the missing data and interpolation the missing value from values that surround it. If you want to fill in missing data using complex techniques, you can even build a machine learning model used to predict missing values. And that's often done in the real world. Let's briefly discuss three techniques that you can use for imputation to fill in missing values. The first of these is hot-deck imputation. You'll first sort all of the records that exist in your dataset based on a criteria of your choice, and for every field that has a missing value, you'll use the value from the previous record. So for each missing value, use the immediately prior available value. This technique is often referred to using the acronym LOCF, which stands for Last Observation Carried Forward. If you're working with time series data, using this LOCF technique is equivalent to assuming that there is no change in your recording since the last time you measured the value for this field. When you use hot-deck imputation using LOCF, the last observation is carried forward to fill in all missing values until all missing fields have been filled in. And this can result in introducing significant bias into your dataset. And because of this tendency to introduce bias, LOCF is not often used in practice. Another imputation technique that you can use to deal with missing data is mean substitution. For each missing value in your dataset, substitute the mean of all of the other available values. The advantage of using this particular technique is the fact that it does not change the mean for that particular variable. However, every technique comes with its own disadvantage. Mean substitution has the effect of weakening correlations that exist between columns of your data. When you use mean substitution, there is no relationship between your imputed variable and other variables that exist in your data. And this can be problematic when you want to perform bivariate analysis to determine relationships that exist between variables. A third imputation technique that you can use to deal with missing values is fit a model. You can fit a regression model to predict missing columns based on other column values. The use of regression imputation has the opposite problem of mean substitution. Regression tends to strengthen correlations between your variables. The variable that you use to predict the missing value will be more strongly correlated with that missing value. Thus, every imputation technique has its own set of trade-offs, regression and mean substitution have complementary strengths and you might want to choose one over the other based on your problem statement.

Identifying and Coping with Outliers

Another common problem that you might encounter when you're working with data in the real world is that of outliers. What is an outlier? A data point that differs significantly from other points in the dataset. Now this outlier might be a genuine data point, in which case it represents information or it could be an erroneous recording. When you're working with outlier data, there are two steps that you need to follow. The first step is to identify outliers that might exist, and the second step is to cope or deal with these outliers. So how would you determine whether a particular data point is an outlier or not. One way is to see the distance of that data point from the mean, or the average, of the dataset as a whole. Now this works well if you're working with univariate data, data that has just one variable. Another way of identifying outliers is to plot all of your data points in some kind of coordinate plane and fit a line on that data. Points that are far away from this fitted line can be considered outliers. Once you've identifier points that are outliers, there are three broad techniques that you can use to deal with this. You can either drop all points so that you ignore outliers altogether. This is what you'd choose to use if you feel that outliers are errors in measurement. You can replace outlier data with a certain maximum or a minimum. You can cap or floor your outlier data points. This will allow all of your points to lie within the range of your choice. And the third way to deal with outliers is to set the outlier data points to the mean, or average, value. Let's first talk about the two techniques that we've discussed here for identifying outliers, distance from the mean and distance from the fitted line. We'll discuss the first of these techniques to start off with, where we use the distance from the average value of the dataset to determine whether a point is an outlier. If you have data in one dimension, the mean or the average is the one number that best represents all of your data points. We simply sum up all of the values and divide by the number of data points. When you're describing data in one dimension, variation of the data is important too. Do the numbers jump around? One way to find out is to use the range, which is simply the maximum value minus the minimum value. When you use range to describe the dispersion in data, the range ignores the mean and is heavily swayed by outliers that might exist. And that's where the variance of your dataset comes in. When you use descriptive statistics on your univariate data, the variance is the second-most important number to summarize the set of data points. How is variance calculated? We'll first need to calculate the mean deviation for our data. For every data point, Xi if you subtract the mean of that data, that gives you the mean deviation. If you compute the square of mean deviation, what you get is the square mean deviation. If you sum up the square mean deviations across your entire dataset and divide by the number of data points that you have, that's when you get the variance. Empirically it has been found that our estimate of the variance for a particular dataset can be improved by tweaking the denominator that you see here, simply use n-1 instead of n, this is called Bessel's Correction, and gives you a better estimate of the

variance. The mean and the variance of your data can be used to succinctly summarize any set of numbers. You know the formulas for each. Now the standard deviation is another term that you might have heard. The standard deviation can be calculated from the variance using a very simple formula. The standard deviation is the square root of the variance of data. These two numbers, the mean and the standard deviation for any data, are often used to identify outliers, points that lie more than 3 standard deviations away from the mean are considered to be outliers. Let's move on to our second technique to identify outliers by measuring the distance from a fitted line. For the sake of simplicity and so that we can represent this in a 2D view, let's consider data in two dimensions. You can see very clearly from this graph that there exists a relationship between X and Y for this data. It's possible to express this relationship using some kind of mathematical formula. Outliers might be data points that do not fit into the same relationship as the rest of the data. So you might imagine that there are points on this coordinate plane that lie far away from the existing points. They do not fit into the same relationship. These points can be considered outliers. We've discussed earlier that there are three techniques that we might use to cope with outliers, drop, cap or floor, or set to mean. Let's consider dropping the data point or setting the data point to the mean. The first step is always to scrutinize the outliers and see why they exist. Do they represent genuine points or are they erroneous observations? If they are erroneous observations, then you'll drop if all of the attributes of that point are erroneous or wrong. If for a particular record all of the fields are wrong observations, there's no point in having that record in there. You'll simply drop the data. Now you might choose to replace a particular data point with the mean, if only one attribute of a record is erroneous. This means that all of the other attributes of that record hold information, you don't want to lose that information. You'll set the particular field, which has an outlier value to be the mean. Now if the outlier data point is not a mistake, it's a genuine legitimate outlier, there are several options open to you. First, you can leave as-is if the model that you're trying to fit on the data is not distorted in any way. If the outliers do not affect your analysis, you'll leave it as-is. If outliers do affect your analysis, you can cap or floor the outlier. We'll first need to standardize the data. Standardizing the data involves subtracting the mean value of the data from all of the data points and dividing by the standard deviation. All of the data points will then be expressed in terms of z-scores, or number of standard deviations away from the mean. We'll cap all positive outliers to be just 3 standard deviations away from the mean, and you'll floor negative outliers to be -3 standard deviations away from the mean.

Summary

And with this, we come to the very end of this module where we understood and studied data cleaning and preparation techniques. We first identified the various problems that hinder analytics, such as duplicate data, missing data, and the presence of outliers. We then discussed common technology tools that we might use to work with data. If you want quick prototyping and something simple to use, you'll work with Microsoft Excel. If you want something that is production ready and works with big data, you'll choose a data warehouse and SQL or Python with Spark on a distributed system. Data in the real world will definitely contain missing fields and values. We discussed two broad techniques, deletion and imputation techniques to deal with missing data. We also discussed how we could deal with outliers and erroneous data. In the next module, we'll see how we can apply the techniques that we just discussed here. We'll see how we can prepare data for analysis using Microsoft Excel and Python.

Preparing Data for Analysis Using Spreadsheets and Python

Module Overview

Hi, and welcome to this module on preparing data for analysis using spreadsheets and Python. In the previous module, we studied the different ways in which bad data can hinder analytics, missing data, duplicates, outlier data, all of these are problems that we need to deal with. In this module, we'll use two different technologies, Microsoft Excel spreadsheets, as well as Python on Azure Notebooks to deal with these problems. We'll first start off working with Microsoft Excel and see how we can use this tool to deal with missing data. We'll specifically look at two techniques, listwise deletion and mean substitution. We'll then move on to identifying outlier data, using Microsoft Excel we'll plot a box plot representation of our data to visualize outliers, and once we've identified them, we'll see how we can use these codes to deal with outliers in Excel. We'll then move on to a different tool and technology to work with data, we'll use Python on Microsoft Azure Notebooks to deal with missing data, as well as outlier data.

Excel: Working with Duplicates and Missing Values

In this demo, we'll see how we can apply some data preprocessing and preparation techniques using Excel spreadsheets. Here we are on a brand new Excel workbook. The workbook is named mobile_price and the sheet that we are working on is called duplicates and missing values. Here we have some data about mobile phones already preloaded into the sheet. This data comes from data that's freely available on kaggle.com at this URL that you see here. This is a dataset that can be used for machine learning and it contains a number of different features of different mobile phones. These are features like battery time, whether it has Bluetooth or not, clock speed, whether it has a dual sim or not, and a number of other details. A simple step in data preparation is to rename columns so that their names are meaningful. Here I'm going to rename the blue column, which represents Bluetooth, to be bluetooth. And I'm going to rename a number of other columns as well. The megapixels of the front camera is fc_megapixel. The depth of the mobile is m_depth. I'm going to rename the column representing the megapixels of the primary camera to be pc_megapixel. Datasets in the real world have to be cleaned before they can be used for analysis. Let's say your data contains duplicates and you want to get rid of these. You can head over to the data menu option on your Excel workbook and, on the ribbon, you'll find an option which says Remove Duplicates. Now this brings up a dialog, which allows you to select in which columns you want to look for these duplicates. All of the columns are selected by default here. I accept these settings and click OK. Excel very helpfully lets me know that there were two rows that were duplicates and Excel got rid of these. That's exactly what I wanted. I hit OK to continue working with my data. Now, I am going to specifically work with the ram column, ram feature in my dataset. Select the Home menu option and here in the ribbon you'll find an option to sort and filter based on the ram. Choose the filter option and this will allow you to filter values. Notice that there is a little icon that shows up, and selecting this icon will bring up the filter dialog for this particular column. By default, all values in this column have been selected. I'm going to deselect select all in order to select specific values. If you scroll down, you'll find an option to select just values which are blank for this column. If you click anywhere outside, this dialog will disappear and this will allow you to see which records contain missing values for ram. When you're using this data for analysis, you may not want missing values in any records. And there are different techniques that you can use to deal with missing values. Here I'm going to select both of these records, right-click, and choose the Delete Row option to get rid of these records altogether. Once I've deleted these records, I can click on the filter icon once again and clear the filter that I've applied on the ram column. Click on Clear Filter and get rid of this filter. This dataset has missing values in other columns as well. I'm going to select the m_depth column, select all of the values within this column, and under the Home menu, go to the Find & Select option on the ribbon. This brings up a little menu, which allows me to go to special, and this brings up a dialog

that allows me to select cells in this column based on the constraints that I specify. I want all of the blank cells in this column to be selected. Blank cells mean that data is missing for this column, and I want to be able to deal with this missing data. Here is a blank cell that is highlighted. Other blank cells in this column will also be highlighted. Another way to deal with missing values in a column is to fill in our values with something meaningful. Let's say you want to fill in zeros for all missing values, and that's exactly what I do here, you can fill this in into one of the cells and hit Ctrl+Enter and all of the blank cells will now have zeros filled in for m_depth. We'll see another technique to deal with missing data here. For that we need to perform some calculations. We'll select the top row of this particular sheet, try click and hit Insert so that we get an additional row about the first row. The mobile_weight column also has missing values and I want to fill in these missing values with the average of the mobile weight. I'm going to calculate the average using an Excel formula. Excel formulas are preceded by an equal to sign and AVERAGE is the name of the Excel function. Once you open up brackets, you can select a particular cell where you want the average calculation to begin and then hit Shift+down arrow key to select all of the cells in this column. Observe the formula bar in Excel, you can see that the average will be calculated for all values from J3 all the way to J2000. Hit Enter to calculate this formula and 141.27 is the average weight of a mobile. We'll now fill in the missing values in the mobile weight column with the average weight of a mobile. Select the column, go to Find & Select, and choose the Go To Special option. This will allow us to select all of the blank cells in this particular column. When you hit OK, you can see that all of the missing values are selected in one go. Now let's fill in a value that we'll use to replace these missing values, that is 141, the average weight of a mobile, hit Ctrl+Enter, and you'll see now that all values that were previously missing now have the value 141.

Excel: Identifying and Eliminating Outliers Using Z-scores

When you work with data in the real world, it might contain outliers, which might be mistakes that you made by recording the data or genuine data. In any case, there are several different techniques that you can apply to work with outliers. And we'll see how you can apply some of these techniques in Excel. Now we're working with the same dataset as before, but on a brand new Excel workbook and a new sheet, and we're using only three columns of the original data, the mobile id, mobile weight, and ram. Let's visualize the data in the mobile weight column, select the column, go to Insert, and in the ribbon you'll find the option to visualize this data in the form of a chart. If you click on this chart dropdown, we'll find an option to create a box and whisker plot. A box plot is a standard visualization technique to see how your data is spread out. The box isn't really very clearly visible in this small view, but these little points that you see at the very top

represent outliers. These are mobile phones in our dataset that are clearly very heavy outliers in their weight. I'm going to make this chart just a little bit bigger so that it's easier to see. The box represents all of the data that are within 25th and 75th percentiles in our dataset. The little vertical lines are whiskers and the horizontal lines that connect to these vertical lines are caps. They represent 1.5 times the interquartile range. Data that does not lie within this range are considered outliers, and you can see that there are three outliers here in this mobile weight box plot. We'll now visualize the ram data in our dataset using the same box plot before. Select the column, go to Insert, select the chart dropdown and choose the box and whisker plot. If you visualize all of the ram details of the mobiles in our dataset, you can see that there's exactly one outlier. This is the ram box plot. I'm going to resize this box plot as well using the knobs provided and once I've done so, I'm going to move it and place it right next to the mobile weight box plot. Here are the two columns in our data which contain outliers. I'm now going to calculate z-scores for the mobile weight column. Z-scores are calculated by subtracting the mean from all of the column values for the mobile weight and dividing by the standard deviation. Z-scores will allow us to express every value in our column in terms of multiples of the standard deviation, and this makes it easier to identify outliers. Anything greater than 3 standard deviations from the mean can be considered an outlier. In order to represent the mobile weight and the ram values using their z-scores, we need to calculate their mean and standard deviations, which I'll do off to the side here. The average or the mean of the mobile weight can be calculated using the average Excel formula. Select all of the values in the mobile weight column to specify as an input to the average Excel function. We've seen earlier that the average weight of a mobile is about 141 grams. The standard deviation of values in a particular column can be obtained using the STDEV function equal to STDEV, and select the mobile weight column once again. This will give us the standard deviation, which is equal to roughly 45 for the mobile weight. Now that we've calculated the mean and standard deviation for the mobile weight column, we can do exactly the same thing for the ram as well. Calculate the average value for the ram column, the average memory is 2128, and in exactly the same way, let's calculate the standard deviation for the ram column as well. And here you can see, the standard deviation is much larger, it is 1097. In order to convert a column into z-scores in Excel, you don't need to manually subtract the mean and divide by the standard deviation, you simply use the standardized Excel function. The first input to the standardized function is the column of values that you want to standardize. This is the mobile weight column, so select it, so that is the B column. The second input is the mean of this column, which is present in cell K7, and the third is the standard deviation, which is present in cell K8. Here is the Excel formula that we use to standardize this data that is converted to z-scores, standardize, range, mean, and standard deviation. You can see that the z-score for a mobile weight of 188 is 1.03, it's

roughly 1 standard deviation away from the mean. I'm going to drag this formula down to fill in the other column values, but before that, I need to ensure that our references to the cells that hold the mean and standard deviation for this particular column do not change when I drag the formula down, which is why I specified KK7, K8. You might already know that the use of the dollar sign in Excel will lock that cell when we drag down the column values. So now let's use that little knob at the corner and drag down this particular cell formula to fill up all of the cells in this column. Another option is to simply double-click on that cell and it will fill in all of the column values. We now have z-scores for the mobile weight. I'll now create a new column of the z-scores of the mobile weight, but this time with the outliers removed. Anything which is greater than 3 standard deviations from the mean, I'm going to consider an outlier. So my outlier range is -3 to 3. Any data point that lies 3 standard deviations on either side of the mean, we consider an outlier. In order to identify and remove outliers, we'll use the if function in Excel. If the z-score is 3 standard deviations away from the mean, we'll replace it with a blank. Notice that we have an or function nested within the if function. The or function basically checks whether a particular z-score is within the outlier range or not. We've specified our outlier range using the z-scores in the cells K10 and L10. If a particular z-score is within 3 standard deviations of the mean, we use that z-score value directly. That is the z-score in the cell D2. If it's beyond 3 standard deviations away from the mean, we simply replace it with a blank. Once we've specified this formula, you can double-click on that little knob in order to fill in this formula for the entire column. If you want to view the outlier values that have now been replaced using blanks, go to Sort & Filter and in this menu, choose the Filter option. This brings up the little filter know on the column header, and use this filter knob, bring up the dialog, and filter by blanks. And you can see that there three values that were outliers and have been replaced using blanks. These were the very heavy mobiles that we had visualized in our box plot. Bring up the little filter dialog once again and hit Select All so that all values are once again displayed in your sheet.

Excel: Clamping Outliers

In this demo, we'll use a slightly different technique to deal with outliers. Let's calculate our z-scores for the ram data first. We've already calculated the mean and standard deviation of the ram values, which means we can use the standardized function in Excel directly. We want to standardize the ram value, so go ahead and select the ram column, and then specify the mean and standard deviation as the input to the standardized function. Remember that z-scores are calculated by subtracting the mean from every value in the dataset and dividing by the standard deviation. We want to use this formula in every cell in the ram z-score column. We want to make

sure that our references to the mean and standard deviation remain unchanged as we copy this formula, which is why we use the dollar sign to lock the mean and standard deviation cells. In order to calculate the z-score for the entire column, select that little knob and double-click and Excel will automatically fill in the formula for the entire column. I'm going to add in another column here named ram zscore clipped. I want to clamp all values that are outliers to be 3 standard deviations away from the mean, in either direction. And one way to do this in Excel is to use two if functions, one nested within another. This nested if function operates on the z-score that we calculated for ram values. It'll first check to see whether a particular z-score is 3 standard deviations away from the mean in 1 direction. If the z-score value is less than -3, it'll simply clamp it to -3. If the z-score value is greater than +3, it'll be clamped to +3. And if the z-score value is within the outlier range, that is 3 standard deviations away from the mean, we use the value as is. Once we've set up the formula for one cell, we simply double-click that little knob at the bottom right of the cell to fill in this formula for all of the cells in this column. Select this column and go to Sort & Filter to filter and view this data. In the menu, select the Filter option. This will set up that little knob that will allow us to filter this data. Clicking on this knob will bring up the filter dialog and select all values which are exactly equal to 3. You can see that there are two mobile phones in our dataset, which are outliers in their ram values. They have very high ram. And they have been clamped to 3 standard deviations. If you want to view all of the rules once again, bring up the filter dialog and choose select all. We saw how we could deal with outliers in two different ways, but how do you know that these outliers have been removed? Let's select a particular column. Here is the mobile weight zscore removed. Go to Insert and let's select the box and whisker plot once again. When you apply this box and whisker plot to the z-scores, you can see that there are no outliers. For the mobile weight column, we had replaced all of the outlier data with blanks. Now let's visualize the ram z-score clipped column, as well, using a box plot. Select the column, go to Insert, go to the charts and choose the box and whisker plot. Once again, this visualization makes it very clear that all of the outlier data have been removed. We've clipped all of the outlier to be 3 standard deviations away from the mean.

Python: Filling Missing Values

In this demo, we'll see how we can clean and prepare our data for analysis using Python. We'll write our Python code on hosted Jupyter notebooks. These are Azure Notebooks on the Azure cloud platform. In order to work with notebooks on the Azure platform, you need to have an Azure subscription, which you can create for free. Azure Notebooks can be accessed at notebooks.azure .com. If you don't already have an Azure subscription, you can use this page to

create one. Or you can go ahead and sign in with your Azure account using the Sign In button on the top right. This will bring up a page where you can specify the username and password for you to sign in. Now I have a personal Azure account and that's the choice I make here. It's not through an organization. And I specify the password and sign in. All of your Azure Notebooks are grouped in logical units called projects. Click on the My Projects link here on top in order to view all of the projects that you've set up. I have one other project here, but I'm going to create a new project to work on the notebooks that I'll use in this course. I call this project preparing-data. Notice that Azure automatically assigns an ID for this project. This ID will be used as a URL prefix for all of the notebooks in your project. You can choose to make these projects public if you want to, if you want others to have a look, but I'll leave that option unchecked. With our project created, let's use this dropdown here to create a new Azure Notebook. A notebook is simply a browser-based shell where you can write code and view results right there on your screen. It's an interactive environment, which we can use to write Python code. You can also use it for other languages, but we'll be working with Python. I'll create a new notebook called FillingMissingValues and I'll use the Python 3.6 kernel. Once you've created the notebook, it will be listed within your project and you can simply double-click on this and it'll open up in a new tab. Here you have it, a Jupyter notebook hosted on the Azure cloud. Your Azure Notebook also comes with all of the common libraries that you might use to process your data, such as NumPy and Pandas. For most of the common libraries, you don't need to install these packages before we use them. In order to understand and implement the different techniques that you can use to fill in missing values using Python, I'm going to set up this toy data frame in Pandas. This toy data frame has just a single column named numbers, and it has some numeric values and some values that are nans, or not a numbers. These are our missing values. Your Pandas data frame makes it very easy for you to check whether any column has missing values. The isnull function on the data frame will check for null or missing values and the sum function will tell you how many records are missing. You can see that our numbers column has 3 missing values. If you want to just get rid of all rules which have field values missing, you can simply call dropna. The dropna function on your data frame will simply get rid of rows with missing values, and if we take a look at the resulting data frame, you'll find that three records from our original dataset have been dropped. These are the records with missing values in their fields. This is exactly equivalent to deleting the corresponding rows in Excel that we saw earlier. Now instead of getting rid of the rows altogether, let's say you want to fill in missing values with a particular default value, such as 0. The fillna function on your Pandas data frame will fill in all missing fields with the value that you've specified here and it happens to be 0. You can also choose to fill in missing values using other data points that are available in that column. One way to do this is to use the bfill or the backfill technique. You can see in the result

that all of the missing values have been filled in with the value 7. So 7 was the next value after all of the missing values and 7 was used to backfill the missing field. In exactly the same way, there is a forwardfill, or an ffill, technique available in Pandas as well. Here, if you take a look at the result, the value 3, which is the value immediately before the missing values start, has been used to forward fill the missing values. In the forwardfill, the value just before the missing value will be considered and that value will be used to fill in the missing data. Another technique to fill in missing values that we encountered in Excel is to fill in missing values for the average of a particular column. Simply call fillna and calculate the average by invoking the mean function on your column. And here in the result, you can see that the missing values have now been filled in by the average value of this column. Data frames also allow you to use statistical interpolation techniques to fill in missing values. Simply call interpolate, and it'll estimate the values that should go in the missing fields. Interpolation is a mathematical technique where you construct new data points within the range of a discrete set of known data points, and that's what Pandas has used here to fill in missing values.

Python: Working with Missing Values on Real World Data

In this demo we performed some data cleaning operations using Python on an Azure Notebook on a real-world dataset. I'm going to upload this data using this upload option available on my Azure project page. These files are present on my local machine so I choose the from computer option and click on this button here to choose files. This will bring up the finger dialog and I'm looking for the mobile_price_classification.csv file. This is a file that we've worked with earlier. Click on the Upload button so that it's uploaded to your Azure Notebook. Click on Done. The file is now available for you to use. Use the dropdown to create a new Azure Notebook to work on this file. I'm going to call this notebook DataCleaningAndPreprocessing, and it'll use Python 3.6. Click on this notebook and it'll open up in a new tab and you're ready to get going. This demo requires the latest version of Pandas available, so make sure that you upgrade the version of Pandas installed on the Azure Notebook using pip install -U pandas. We'll also use the seaborn library for visualization, so pip install the latest version of seaborn as well. Set up the import statements for the Python packages that we plan to use and then let's go ahead and read in the contents of the csv file that we just uploaded using pd.read_csv. We're already familiar with this mobile dataset, it contains the features of a number of different mobile phones. If you scroll all the way to the right here, you'll see that at the very end we have a column called price_range. All of these mobile phones in our dataset have been categorized or bucketed into four price categories, low, medium-low, medium-high, and high, represented by the numbers 0, 1, 2, 3. Let's

quickly explore this data that we have. The shape of a data frame will give you the number of columns and records in this dataset. This dataset contains 2002 rows and 22 columns of data. The columns attribute in a data frame lists out the names of all of the columns that we are working with. You can see that all of these columns, except for the id column, represent features of mobile phones. So I'm going to go ahead and drop the id column so that we can work with the remaining 21 columns. The describe function on a Pandas data frame gives you a quick statistical overview of all of the numeric columns of data present in your dataset. This allows you to get a quick feel for your data, the number of records, the mean and standard deviation of every column, the min, max, and the quartile ranges. Let's now perform some data cleaning operations, many of these operations we've done before in Python. The first thing we'll do is to rename some of the columns so that their names are more meaningful. We'll rename the blue column to be bluetooth, fc and pc to fc_megapixel and pc_mexapixel, and m_dep to m_depth. Pandas data frames offer this very helpful function called duplicated, which allows us to check whether any of the records in this data frame are duplicates. The dupes variable contains a list of true false values, indicating whether a particular column is a duplicate or not, in order to see how many of these columns are duplicates, can pass in the dupes column to the built-in sum function in Python. And you can see that there are two records which are duplicates. Let's get rid of this duplicate data by calling data.drop_duplicates. If you take a look at the resulting data frame, you can see that the two duplicate records have ben dropped. We now have just 2000 records and every record has 21 columns. Let's now see which of these columns have missing values. Call data.isnull .sum and this will list out all of the columns and the number of missing values in each column. You can see that there are several columns here with missing values. Another way to detect missing values using Pandas data frame is to call isna and sum. Now this is a new function that has been added and may not be available in older versions of Pandas. This gives you the same result as isnull.sum. Let's now deal with these missing field values using techniques that we've studied earlier. In the fc_megapixel field, I want to fill up all missing values with 0. Now you can see that fc_megapixel contains no missing values. Before we deal with the missing values in the ram column, let's see how many unique values are present in this column by calling the unique function on this column. You can see that there are 1562 unique values. Note here that missing values are also considered unique. Now I'm going to use the fillna method, backfill, or bfill, method to fill in these missing values. Once the missing data for the ram values have been filled in using backfill, if you take a look at the number of unique values, you'll see that it's 1561. The missing field that was considered a separate unique value has now been backfilled with one of the existing data points. Which is why the number of unique values has reduced by 1. Once again, let's check which fields have missing values by calling isnull.sum. You can see that the ram field no

longer has any values missing, thanks to our backfill. Now let's fill in the missing values in the mobile weight column by using the median of the mobile weight. Calculate the medium for this column by invoking the median function and use the fillna to fill in the missing values with medians. The mobile weight column now should no longer have any missing values, and you can confirm this by calling isnull and sum. Now whatever other fields contain missing values, I want to just drop those records. The dropna function in Pandas will drop all records which have any fields missing. Our dataset now should have no missing values. Call isnull.sum, and you can see that the number of missing values are 0. Now we are left with fewer records than earlier in this dataset, just 1995 records. This is because we deduped and then dropped some of the records that had missing fields. At this point, our dataset is completely clean. Here are the columns in our dataset. I'm going to write this out as a csv file called mobile_data_cleaned.csv. Index is equal to false will not include the index of the data frame in the csv file. You can run the ls command here within your Azure Notebook and you can see that mobile_data_cleaned.csv is present in the current working directory.

Python: Identifying and Removing Outliers

Now that we have a cleaned dataset, let's see how we can deal with outliers in our data. Now I'm going to extract all of the numeric features from my original data frame into a separate data frame called numeric data. So mobile features, such as bluetooth, dual_sim, wifi, price_range are categorical values, they are not numeric values, so I'm going to go ahead and drop them. So numeric data contains only numeric features. We'll now examine these numeric values for outliers and then deal with those outliers. I'm going to extract all of those columns with categorical data, that is data with discrete values, into a separate data frame called categorical_data. If you take a look at the categorical data, you can see that we use 0 1 values to indicate the presence or absence of a feature on a mobile phone, and the price range contains discrete labels, indicating whether a mobile is low priced, medium priced, or high priced. In order to visualize the outliers in our numeric data, I'm going to import the matplotlib and seaborn visualization packages. Within our mobile dataset, we know that there are outliers in the ram value. So let's visualize the ram values of all of the mobile phones in a box plot. The box plot is a very useful tool to visualize outliers. The center line here represents the average ram value. The box here represents the interquartile range, that is the values at the 25th and 75th percentile. The whiskers here represent 1.5 times the interquartile range away from the mean, and this little dot that you see on top here represents the outlier data. We note that there are certain mobiles that are outliers in the mobile weight as well. Let's visualize this using a box plot. And you can see the outliers represented here

on top. There are three mobiles represented in our dataset that are indeed very heavy. So far we've seen how we can visualize individual column values in box plots. With Pandas, you can visualize all of the columns in one go in a single box plot. We pass in all of the numeric features as input to this box plot and we arrange the xticklabels to be at 90 degrees so that they're clearly visible. The column names will be the xticklabels. You can see the range of all of the numeric features in a single box plot. You can see that ram values tend to be spread out over a larger range, whereas battery power tends to be clustered together in a smaller range. Now in order to detect outlier data, we are going to standardize our dataset by calculating z-scores. This can be done very easily using the StandardScaler preprocessing object from the scikit-learn library. Instantiate the StandardScaler and call fit_transform on the numeric data. Fit transform will scale all of our numeric data to have zero mean and unit variance. All of the numeric features will be expressed in terms of their z-scores, based on how many standard deviations they are away from the mean. This operation might result in a warning. You can ignore this. This warning just tells us that some of the columns that were originally integer values have been converted to floating point values, that's expected when we perform the scaling. I'm now going to create a new data frame containing all of the scaled values. The columns of this data frame are the same columns as our original numeric data. Here are all of the features in our original dataset, battery power, clock speed, front camera, back camera, megapixels, all expressed in terms of z-scores. If you call describe on this new data frame of scaled_data that we just created, you can see that the mean values for all of the numeric features are very close to 0 and their standard deviations are very close to 1. Standardizing your dataset makes it much easier to compare data with different ranges. Let's plot a boxplot of the standardized data, and in this resulting visualization, you can see that all of the boxes for all of the features have their center lines at 0, indicating that their mean is 0. Values are expressed in terms of z-scores. This makes it much easier for us to identify outliers in our data. Now that we've seen that our data contains outliers in certain columns, let's see how we can remove them. For this, here I'm going to consider any data point that is outside of 1.5 times the interquartile range from either the 25th or the 75th percentile as an outlier. For this, I need to calculate Q1 and Q2, which is the 25th percentile and the 75th percentile of my data. I use quantile function in Pandas for this and I store the resultant Q3 and Q1, the interquartile range is Q3 minus Q1. If you print out IQR, you can see the interquartile range for all of the numeric features that are present in our mobile dataset. Now I'm going to get rid of all records which contain outlier data because I consider those to be errors. And here is how I perform this conditional check for outliers using Pandas. From all of the numeric features in the original numeric data frame, I try to find those values that are 1.5 times the interquartile range, away from Q1 or Q3. The tilde sign that you see here at the beginning is what gets rid of all of the

records that satisfy this outlier condition. If you look at the data once outliers have been removed, you can see that we are left with just 1971 records from the original dataset. We can confirm that outliers have indeed been removed by taking a look at the boxplot of the outliers removed data. If you look, all of the boxplots remain more or less the same, but there are no outliers plotted. We've successfully identified and removed all of the outliers present in our dataset.

Python: Training an ML Classifier Using the Cleaned Dataset

Now that we've cleaned and prepared our dataset, it has no missing values and we've also standardized all of the numeric features, let's fit a little machine learning model to see how this clean dataset can be used. We'll build a little classification model to categorize our mobiles as high priced, medium priced, or low priced. We'll use the train_test_split function from the scikit-learn library to split our data into training data and test data. We won't use raw values for our numeric features, we'll use the scaled data that we had set up in the scaled_data data frame earlier. We set the index of the scaled_data and we set the index of the categorical_data as well. Once we reset the index values, we can then concatenate our scaled data and categorical data to get a final data frame which contains all of the features that we are going to feed into our machine learning model. The data frame contains the scaled numeric features for all of the mobile phones, and if you scroll over to the right, all of the categorical features are also available in this data frame. This is the cleaned dataset that we working with. If you call isnull and sum, you can see that there are no missing values in this data. The X values, or features, that we'll use to train our classification models, we'll store in the X variables. This includes all of the columns except the price_range. The price range forms the categories, or the Y values, that we'll try to predict with our classification model. We'll now use train_test_split to split our X and Y values into training data that we'll use to train our machine learning model and test data that we'll use to evaluate our machine learning model. Eighty percent of the original dataset, that is around 1600 records, we'll use to train our model, and the remaining 400 records to test our model. We'll build a simple LogisticRegression classifier model, import the LogisticRegression estimator object from sklearn.linear_model. Instantiate this estimator, the solver is the optimization algorithm that this estimator uses, which is the lbfgs solver. Multi_class is equal to multinomial indicates that we want to categorize our input data into more than two categories. Mobile phones are categorized into four output categories, low, medium-low, medium-high, and high. And this classifier model will train for a maximum of 10, 000 iterations. We'll kickstart the training process by calling logistic_model.fit on the training data, pass in the y_train values as well, and once we have a fully trainer classifier, we can use logistic_model.score to evaluate how well this model performs on

the test data. You can see that our little ml classifier here did very well with 93.7 percent accuracy in predicting test values. Now before we move on, I'm going to switch back to the mean project view of my Azure Notebooks and download this mobile_data_cleaned.csv file. You can select this download option here and save this file onto your local machine. I'll now have this clean dataset to use elsewhere in my demos.

Summary

And this demo brings us to the very end of this module where we saw how we could prepare data for analysis using Microsoft Excel spreadsheets, as well as Python. Having understood the techniques to deal with missing and outlier data in a previous model, we used this module to get some hands on practice. We first worked with Microsoft Excel spreadsheets and saw how we could deal with missing data using listwise deletion and mean substitution. We then used box plots to visualize and identify outliers that exist in our data. We converted data with outliers to z-scores using the standardized function in Excel. Standardize allows us to represent our data in terms of standard deviations. We considered all data above and below 3 standard deviations from the mean to be outliers. We then either deleted these outliers or capped and floored their values. And finally, from Excel, we moved on to using the Python programming language to clean and prepare data. We dealt with missing data, as well as outlier data, writing Python code on Azure Notebooks. In the next module, we'll study how we can collect and process data to extract insights. We study a few standard data preprocessing techniques and discuss how we can scale to meet the needs of big data processing.

Collecting Data to Extract Insights

Module Overview

Hi, and welcome to this module on Collecting Data to Extract Insights. In this module, we'll discuss the different sources that you can use to access data. And we'll then study a number of data preparation techniques. We'll discuss techniques such as standardization and normalization that can be used to preprocess numeric features of our data. We'll also discuss how we can use binning and sampling to get a representative sample of our dataset. We'll then move on to

discussing the processing techniques that you would use to work with big data, that may not sit in memory or a single database. We'll discuss different kinds of data that you might have to deal with, batch data or streaming real time data. When you're working with streaming data, time is significant, and we'll discuss the concept of event time and processing time. A data professional helps make business decisions based on data, and we discussed earlier the two hats of a data professional. The first is to find the dots, identify important elements in a dataset. This involves careful data collection and applying techniques to deal with missing data, outlier data, and other problems. The next step is to connect the dots, explain these elements that we've identified in our data via relationships with other elements. There are three important steps involved in connecting the dots. The first is to process the data so that we can use it for analysis in models. The second step is to build and refine these models that we'll use for prediction, and finally, incorporating real-world data into these models. Each of these categories encompass a vast area into which there has been a lot of research, a lot of study. The middle step, building and refining models, involve machine learning techniques and other predictive analysis, that is beyond the scope of this particular course. We'll focus on the remaining two steps, processing data for use in models and incorporating real-world data into models.

Standardization

When you're working with data, standardization is a common preprocessing technique that you'll use with numeric features. Let's consider your dataset represented in a tabular format. There are key features of columns in your data and there are a total of n records. So your data could be represented using this matrix or a tabular format. If you explore the numeric values in your dataset, you'll find that different features will have different means, different ranges, different standard deviations. So it's hard for you to compare these features. When you preprocess your data using standardization, this helps build more robust machine learning models. Standardization involves first calculating the average of each column of data. Next, for each of these columns, you'll calculate the standard deviation of that data. So you have averages, X1 through Xk, and standard deviations X1 through Xk as well. When you standardize your data, from every value in your dataset, you'll subtract the average of that column and divide by the standard deviation. Once you've performed this mathematical operation, each column of the standardized data has a mean or an average of 0 and a variance or a standard deviation of 1. Standardization involves expressing all of your data in the form of z-scores, or in terms of how many standard deviations away from the mean. Standardization is a column wise or feature wise operation. It operates column-by- column and yields features with zero mean and unit variance. So the

features of your original data was distributed like the visualization you see on the left, the means are different, the standard deviations are different. After you've standardized your data, all of the means will be the same, equal to 0, and data will be expressed in terms of its standard deviation. The technique that we just discussed involves the use of the mean and the standard deviation. Mean, as you know, is a measure of central tendency and standard deviation is a measure of dispersion. The mean value of a dataset tends to be very sensitive to the presence of outlier data, which is why if you want a more robust way to standardize your data, you'd use robust standardization. An alternative measure of central tendency, that is robust in the presence of outliers, is the median and the interquartile range is also a measure of dispersion. Robust standardization involves subtracting the median of your data from every value and dividing by the interquartile range. This technique is called robust standardization because the output does not change much due to the presence of outliers. The objective of both of these techniques is exactly the same, represent all of your numeric features such that they're comparable, and use this preprocessed data to build more robust ML models. If your original data can be expressed using a visualization, such as the one that you see on the left, where you can see that the range of the data is very different and there are outliers, after robust standardization, you'll get data that is distributed like you see on the right. The median of all of your features, after robust standardization, will be equal to 0.

Normalization

Another common data preparation technique that you'll use with numeric data is normalization. And we'll study that here in this clip. Every record in your dataset will be made up of a number of different fields. It can be one field or many. So every record can be thought of as an input vector. Normalization is the process of scaling these input vectors individually to unit norm, or unit magnitude. Normalization is commonly applied to data that you feed into an ML model, often in order to simplify cosine similarity calculations. Often in machine learning techniques, you need to know how similar two data points are, how similar those vectors are. Cosine similarity is a measure of similarity because two non-zero vectors. This is widely used in ML algorithms, especially in document modeling applications. Let's understand intuitively how cosine similarity works. Consider two orthogonal vectors, A and B, which are at 90 degrees to one another. Orthogonal vectors represent uncorrelated data, so A and B are unrelated or independent. The cosine similarity of orthogonal vectors is the course of 90 degrees, which is equal to 0. You know that a cosine similarity of 0 means that A and B are unrelated. Let's consider two vectors that are aligned, vectors A and B represented in this diagram are parallel to one another and the angle

between them is exactly equal to 0 degrees. These vectors can be considered to be perfectly aligned and the data points A and B have a correlation of 1, the highest possible correlation that can exist. So if you think of the cosine of 0 degrees, you know that it's equal to 1. After orthogonal and aligned vectors, let's consider opposite vectors. Vectors A and B here point in opposite directions, so the angle between these vectors is 180 degrees. Such vectors are said to be perfectly opposed. The correlation between these two vectors is equal to -1, which is the lowest possible. And if you calculate the cosine of the angle between these two vectors, that is cosine of 180 degrees, it's equal to -1. It's pretty clear that the cosine similarity is a quick and intuitive way to express the alignment that exists between two vectors to see how similar two vectors are. Each vector here can be thought as representing a single point in our data. If you have three dimensions, a point can be represented using the coordinates X, Y, and Z. So now we understand why cosine similarity is useful. We know that many ML algorithms use cosine similarity to calculate similarity between vectors, but how does normalization fit in? If the angle between two vectors is represented by theta, the cosine similarity between these vectors, that is cos of theta, is given by this formula you see here on screen. In the numerator, we have the dot product of the two vectors, A.B, and in the denominator we have magnitude of A multiplied by magnitude of B. The formula for the square of the magnitude of vectors is as you see here on screen, we have magnitude of A squared and B squared. If you look at this mathematical formula, you know that calculating the cosine similarity would be reduced to the simple dot product, A.B, if the magnitude of each vector were equal to 1. And this is exactly what normalization does. We pre-convert the vectors, A and B, to unit norm vectors to simplify cosine similarity calculation. Unit norm vectors are obtained when you divide the vector A by its magnitude using the formula that you see here on screen, and the vector B by its magnitude using the same formula. When you pre-convert A and B to unit norm vectors, calculating the cosine similarity becomes very straightforward and this greatly improves the performance of your ML models. One thing to note about normalization, normalizing is a row-wise operation, it applies to a vector at a time, whereas scaling and standardization that we saw earlier is a column-wise operation. Normalization is a general technique of converting a vector to unit norm. There are three different norms defined for vectors, the L1 norm where the sum of the absolute values of the components of the vector is equal to 1, the L2 norm is the traditional definition of vector magnitude that we saw earlier. L2 norm is where the square root of the sum of the squares of the vector coordinates is equal to 1. The third kind of norm that can be used is the max norm. This is where the largest absolute value of the elements of a vector is set to 1 and other elements are expressed in terms of this largest value.

Binning

When you're preparing and processing data, it's possible that certain numeric features may be in the form of a continuous variable. You might want to convert these to categorical form before processing, and that's exactly what binning does. Data binarization is a specific binning technique. This converts a continuous variable, which can take on any numeric value within the range, into a binary categorical variable. Categorical data refers to data which has discrete values. Binary categorical data can have any of two values. Continuous data can be binarized based on a threshold, which is specified by us, the user. Let's visualize how binarizing data works. Let's say you have continuous data that can be represented in two dimensions. You'll draw a line through this data, which represents the threshold. The threshold can be thought of as a dividing line for your data. All points which lie above the threshold can have one value, let's say it's 1. Points that lie below the threshold will have another value, let's say it's 0. The objective of binarizing data is to convert a continuous input to a binary categorical output. But what if you want to divide data into more than two categories? This brings us to discretizing data. This generalizes the idea of binarization and converts continuous data into categorical data arranged into a specified number of bins, and the number of bins can be two or more. Let's understand this concept with a visualization. Let's say your continuous data looks something like this. A continuous string of data is fed as the input and after you discretize or bin the data, you might have three bins. Discretization is a process that the original continuous data is converted to discrete categories and the output categories can be two or more. There are several different techniques or strategies that you can use to bin your data. Let's consider just three of these. The first is uniform binning where the bin widths are constant in each feature. Say the numeric values for your feature range from 0 to 100 and you've chosen a bin width of 10, you can convert your continuous data to 10 discrete values corresponding to your bins. You can also choose to bin your continuous data using quantiles. Here is where all bins in each feature will have approximately the same number of samples or data points. Or, you can choose to apply some kind of clustering technique to your data and bin your data based on the centroids of a K-means clustering procedure. Points that are close to one another will belong to the same bin and will have the same discrete output.

Transactional and Analytical Processing

Data that we have to work with and process nowadays is big data, data that is gigabytes, terabytes, or petabytes in size. We had spoken earlier about the essential steps in connecting the dots to analyze data. We need to process data for use in models and incorporate real-world data into models. We dove briefly into data processing techniques earlier. We'll now talk about

incorporating real-world data into models, and we know that data in the real world tends to be big. When you work with data, there are two kinds of processing operations that you can perform, transaction and analytical processing. Consider John who works in order management support. He is responsible for tracking and delivering ecommerce orders on time. Anna, on the other hand, is a revenue analyst. She is responsible for tracking and monitoring revenues. The problems that John deals with tend to be urgent and require immediate action. Let's say deliveries in Kent, Washington are delayed because the courier company has had a computer outage. John might have to go in and assign all of these orders to another courier company which services the same region. As a revenue analyst, Anna will have different kinds of problems that she has to deal with. Her manager might want an update on last month's revenues and last month was an unusually slow one. Anna might pull up data which spanned over the last 5 years, to check whether the ecommerce site has experienced any seasonal variation in demand. And herein lies the differences in how we process the data. John will perform transactional processing of data, Anna will require analytical processing. Let's compare and contrast these two types of processing side by side. Transactional processing involves ensuring the correctness of individual entries in the data, whereas analytical processing involves analyzing large batches of data. For transactional processing, recent data is more important or significant, access to recent data from the last few hours or days is needed. Analytical processing involves accessing large amounts of data spanning months or even years. Transactional processing typically involves updates of the data in real time. Analytical processing mostly involves reading the data to extract insights. Transactional processing systems need to enable fast real-time access to data, whereas analytical processing systems need to be able to handle long running jobs. Typically for transactional processing, all of the data is available from a single source. For analytical processing, to extract insights, you might require multiple data sources. Back in the day, both transactional, as well as analytical processing objectives, could be achieved using the same database system. Data wasn't really very large. It could fit on a single machine with backup. Data tended to be structured and well defined and it's important that you be able to access individual records in your data. There was no replication and updated data was available immediately. But as the size and scale of data has grown, it's very hard to meet all of the requirements for transactional, as well as analytical processing, using the same database system. Big data is no named because it's huge. Data does not fit on a single machine or system, it has to be distributed on a cluster with multiple machines. Big data need not always be structured or well defined. It can be semi-structured or unstructured data. Typically, when you're working with big data, you don't need random access to data. You simple need to be able to process a huge amount of data to extra insights. Big data is typically used for read operations, data is replicated, which means that propagation of updates take time, as updates

have to be propagated to all replicas. When you're working with very large data, you're likely to have multiple sources from where you ingest data. Different sources may have different unknown formats. Now that we've understand the requirements for transactional and analytical processing, let's consider the tools that we would use for these. Transaction processing is typically performed using traditional relational databases. This is typically a SQL database hosted on-premises or on a cloud platform. Analytical processing, on the other hand, is performed using data warehouses.

Vertical and Horizontal Scaling

While working with big data, people tend to talk about the 3 Vs big data. The 3 Vs represent specific attributes that define big data. The first of these Vs refer to volume. This is the amount of data that has to be ingested and processed by your system. Volume, as you might imagine, is the main characteristic that makes data big. Data is no longer measured in megabytes. Gigabytes, terabytes, or petabytes are the scales that we refer to. The second V is the characteristics of big data refers to variety. Now data is not just structured data that can be stored in a relational database, it involves unstructured data, such as tweets, texts, images, videos. All of these can be ingested and processed using big data technologies. The variety refers to the sheer number and type of sources available. And finally, the third V that characterizes big data is the velocity. Velocity refers to how fast data comes in that needs to be processed. Batch data is data that you have stored on disk somewhere, streaming data is real-time data that's constantly coming in and needs to be processed. You might hear of two other Vs in reference to big data. The first of these is veracity. Can your data be trusted? And the last V can be thought of as value. Is your data useful? Can you get insights that help you make business decisions? It's pretty obvious that you need to scale to meet the needs of big data. There are two types of scaling possible, vertical and horizontal. As the size and scale of the data that you're working with grows, vertical scaling involves making your processing machine more powerful. You continue working with data on a single server, but you make that server more powerful with more memory, more CPU, more disk space. Vertical scaling, of course, has its limits. There is a limit to how powerful a single machine can be. Horizontal scaling, on the other hand, as data size grows, you add more servers. A single server by itself may not be very powerful, it may even be made of off the shelf components. However, all of these servers together can process huge amounts of data. And horizontal scaling using a cluster of machines is what the world is moving towards. Vertical scaling involves the use of monolithic architecture. Horizontal scaling uses a distributed architecture where we work on a cluster of machines. Vertically scaled systems do not require orchestration. All of the data is stored on a single machine. Whereas when you work with horizontally scaled system, you need to

have orchestration software that distributes the data and the processing. Vertical scaling does not require you to shard data and spread them across machines, whereas horizontal scaling does. With vertical scaling, you might back up data, but your data is not replicated in real time, whereas with horizontal scaling, in order to keep your data secure, it needs to be replicated. With vertical scaling, since all your data is stored on a single machine, it's easies to ensure the consistency of your data. Whenever you perform read operations, all of the updates that have gone before it will be reflected in your data. For horizontally scaled systems, with replication it's hard to ensure consistency. Vertically scaled systems can offer strong consistency fairly easily, whereas horizontally scaled systems offer only eventual consistency. We'll talk more about this in just a bit. Usually vertically scaled systems offer transactional support, atomicity, consistency, isolation, and durability, the ACID properties. With horizontal scaling, ACID is hard to provide. Vertically scaled systems are typically used for transactional processing or OLTP. Horizontally scaled systems are used in OLAP applications for big data, where OLAP stands for online analytical processing. Before we round out our discussion of scaling to meet big data needs, let's talk about strong consistency and eventual consistency. Vertically scaled systems which have a single powerful machine can perform strong consistency guarantees, whereas horizontally scaled systems only provide eventual consistency guarantees. What does this mean? Well let's consider a single server that you see on the left. You receive an update request for your data, data is immediately updated on that server and the request is returned to the user. Data needs to be updated in exactly one place, all other requests for that data will receive the latest updates, that is strong consistency. Let's say you have a distributed system, horizontally scaled one with multiple servers, you receive an update request for your data, the master node of the data will be updated, but this update might be propagated to replicas later on. It won't be immediately replicated. The request will return first. The replica will be updated later. Now before the replica updates, if a request for this data is received on the replica, you might receive stale contents in the response. Data will eventually be consistent, not immediately.

Batch and Streaming Data

Nowadays all of the data that you need to process may not be available on a disk or a database somewhere. You might have to ingest data in real time, that is streaming data. Batched processing is typically applied to datasets that are bounded in nature. Bounded datasets are processed in batches using long running jobs. With batch processing, all of the data that you need to work with is available up front. On the other hand, you may be ingesting data in real time, unbounded datasets are processed as streams. You'll never have the entire dataset available at

any point in time. You'll have to work with and process data as it comes in. Let's quickly compare and contrast batch versus stream processing. Batch works on bounded, finite datasets where all of the data is available up front. Stream works on unbounded, infinite datasets where data is processed in real time. Batch is typically a slow pipeline from data ingestion to analysis, data might first be stored in some database or disk somewhere. With streaming data, on the other hand, the processing is immediate. As soon as data is received, it's processed. When you're processing batch data, it's typically in the form of long running jobs. You'll get periodic updates as jobs complete. With streaming data, processing is continuous. You'll receive updates continuously as jobs run constantly. With batch data, the order in which the data was originally received is not significant, neither is it important. When you're working with batch data, since all of the data is available up front, you can always rearrange it to be in any order that you wish it to be. Streaming data typically deals with events and every event has an associated time. The order of the data is important and out of order arrival of data is often tracked. Since all of the data is available up front before processing, batch data has a single global state of the world at any point in time. With streaming data, there is no global state, only the history of all of the events that have been received by the system. Streaming data often refers to real time events or activities. Data is received in the form of a stream. This can be log messages, tweets, or climate sensor data. All of these are examples of streaming data where events or information is received in real time. There are different techniques that we can use to process streaming data. You can process the data one entity at a time, such as applying filtering operations to get error messages, finding references to the latest movies, or tracking weather patterns. Once you've processed the messages that you've received in a stream, you can store, display, or act on these filtered messages. You can choose to trigger an alert based on your logs, you can show trending graphs based on what movies or shows are currently popular, or you can use climate data to warn of sudden squalls. Stream processing techniques also involve working on windows of data. You aggregate all of the events that occur within a particular window and perform some kind of analysis or operation. Batch data is typically stored in traditional storage, such as files or databases. These files and databases are the source of truth for that data. When you work with streaming data, you have a stream-first architecture. Your stream can originate from data stored in files, databases, or real time streams. All of these are typically buffered and passed in to a message transport system. And from the message transport, they are passed on for stream processing. When you have a stream-first architecture, the stream itself, that is the message transport, is the source of truth for your data.

Event Time, Ingestion Time, and Processing Time

When you're working with streaming data and stream processing systems, the notion of time becomes extremely important. And there are several different notions of time that you need to understand. The first of these is event time. This is the time at which the original event occurred that was streamed into our stream processing system. The second bit of time that's associated with the streaming event is ingestion time. This is the time at which that event was ingested into our stream processing system. And finally, every streaming event is associated with a third notion of time, that is processing time. This is the system time of the machine that processes these streaming entities. Event time appears first in chronological order. This is the time at which the event occurred at its original source. The original source of the streaming data might be a mobile phone, a sensor, or a website. The time at which the original event occurred at this source is the event time, and this is usually embedded within your streaming records. It's not possible for the stream processing system to know about event time unless it's available within the events that are streamed in. Event time is significant and important because it gives you correct results, even when data appears out of order at the stream processing event. Let's say you have a streaming event that has to travel a long way from a distant part of the world, event time will tell you exactly when it occurred. If you're working with streaming data, ingestion time is the time when that event enters your stream processing system. This is a timestamp that is assigned to an event by a stream processing system and is chronologically after the event time. It's important for you to realize that ingestion time does not give you the correct order in which events occurred. It only tells you when these events were first seen and ingested by your system. Once your streaming data has been ingested, they might be buffered before they are processed. The processing time refers to the system time of the machine that processes these streaming entities. Processing time is chronologically after the event time and the ingestion time, and this typically tends to be non-deterministic in nature. You have no idea how long a particular event will remain in a buffer before it's processed. Working with processing time is far simpler than working with event time because it's simple. There is no coordination required between streams and the processors of streaming data.

Summary

And with this, we come to the very end of this module where we saw how we could collect and process data to extract insights. We first studied a number of different data pre-processing techniques to get data into a form that we can use for analysis, or to feed into an ML model. We studied standardization and normalization of data. Both of these are techniques used to pre-process numeric data and when applied to training data can help build more robust ML models.

We then studied binning and sampling techniques, which allows us to convert continuous data to categorical form. We then discussed the scale and processing requirements to work with big data. We studied and understood the differences between transaction processing and analytical processing of data. We then moved on to discuss the different kinds of data that we might want to process, batch data and streaming real time data. We compared and contrasted batch processing, as well as stream processing systems. And finally, we discussed the notion of time that is extremely important in the case of stream processing. We studied event time, ingestion time, and the processing time of streaming events. In the next module, we'll see how we can load data from different sources into a SQL database and process data using the structured query language.

Loading and Processing Data Using Relational Databases

Module Overview

Hi, and welcome to this module on Loading and Processing Data Using Relational Databases. In this module, we'll work with the Microsoft Azure cloud platform and you'll need an Azure subscription in order to perform the demos of this module. We'll load and work with SQL data on the cloud using the Azure SQL Database managed service. Once we've loaded data into a relational database, we'll run SQL queries using the Query Editor available on the Azure portal. This is a browser-based query editor and it requires no additional installation or tools. We'll then install SQL Server Management Studio on our local Windows machine and use this tool to connect to our Azure database on the cloud. And another technology that we'll use is the Azure Data Factory. We'll see how we can use pipelines within the Azure Data Factory to load data from different sources. All of this data will be loaded into our Azure SQL Database. Working on the Microsoft Azure platform is seamless. Anything that you can do on your local machine, you can do on the cloud. The Azure SQL Database is a managed relational database service on the Microsoft Azure cloud platform ideal for transaction processing applications. This offers the same SQL server database that you might be familiar with working on your local machine, but as a Platform-as-a-Service, or PaaS, offering. The Azure SQL Database is predictable and works

exactly like SQL Server because it shares the codebase with Microsoft SQL Server database engine. As more and more companies migrate their infrastructure to the cloud, Microsoft is now cloud-first. New features that are added to the SQL Database go to the Azure SQL Database on the cloud before they are added to local machine installations. Azure SQL Database is a fully managed relational database on the cloud. All of the administration and configuration requirements are mostly taken care of by Microsoft. There is no overhead for patching or upgrading your system. The Azure cloud platform offers a number of deployment options for your SQL Database. You can deploy a single database that comes with its own set of resources, managed via a SQL Database server. A single database is isolated from other databases and from the instance of SQL Server that hosts the database. This is the deployment option that we'll choose in our demos. Another option is to deploy your Azure SQL Database in an elastic pool. An elastic pool is just a collection of databases with a shared set of resources, managed via a SQL Database server. Single databases can be moved into and out of an elastic pool. Or you can choose to deploy your SQL database as a managed instance, which is a collection of system and user databases with a shared set of resources. In this module, we'll connect and work with our Azure SQL Database on the cloud using SQL Server Management Studio. SSMS is available only for your Windows machine. It has been around for a really long time and is an old favorite of database administrators. SSMS has now been upgraded to work with all SQL services, including SQL Server, the Azure SQL Database, and the Azure SQL Data Warehouse. In addition to SSMS, another tool that we'll use on the cloud is the Azure Data Factory. This is a fully managed service meant for building complex, hybrid ETL, or extract transform load pipelines that serve to bring together data from disparate sources, thus integrating data silos. In addition to copying data from source to destination, Azure Data Factory pipelines can include Hadoop and machine learning transformations on your data.

Creating an Azure SQL Database

In this demo, we'll create and set up an Azure SQL Database on the Azure cloud platform to work with relational data. The main portal for the Azure cloud platform is accessible at portal.azure .com. This is the main dashboard which gives you an entry point to all of the services and resources that Azure has to offer. On the left hand pane, you can expand this little arrow button here and this brings up a navigation pane, giving you access to all of Azure's resources. This is where you have the link to create a new resource as well, and that's what we'll click now in order to create a new SQL Database. Go to the Databases option here and on the right-hand side, choose SQL Database. This is the single database deployment option on Azure, which creates a

database in Azure SQL Database with its own set of resources, and it's managed via a SQL Database server. Every resource that you create on the Azure cloud is part of a resource group. A resource group is simply a logical group of resources. I'm going to create a new resource group called loonycorn-rg in order to hold the SQL Database. You'll need to name this database. I'm going to simply call it loonycorndb. And I'm going to name the server for this database. Let's create a new server here, we don't have any existing servers. And I'm going to call the server loony. The full name for the server is loony.database .windows .net. The next step is to specify a server administration login. This is the loony user. Go ahead and specify a password, confirm this password, specify where you want this database to be located, this is East US for us, and hit Select. East US is the geographical region where this database resource will be located. Once you've created a database, you have the option to configure your database as well. Click on this Configure databases link, and you can specify the number of DTUs and the maximum size of data that you can store here. A DTU on Azure is a database transaction unit, which is a blended measure of CPU, memory, and data input/output operations for your SQL Database. We'll accept the default size of 10 DTUs, but we'll change the data max size to be just 500 MB, we don't want to store more than 500 MB of data. Based on your configuration settings, you'll pay a different amount per month for your database. Here is the estimated cost per month for this particular set of configuration settings. Go ahead and hit Apply to accept these configuration settings and let's continue with the creation of this Azure SQL Database. Click on Review + create. This will take you to a page where you can review the configuration settings for your database. Click on the Create button here and wait for this resource to be deployed. Once you get the cheery message that your database has been deployed, you can click on this button to go to your database resource. We'll now see how we can use this Azure SQL Database on the cloud. On the left pane, you'll find a number of useful links pertaining to this Azure SQL Database, and the one that we are going to use right now is the query editor. This allows us to write our SQL queries right here within the browser. You'll need to log in to the database in order to use the query editor. You can use the SQL Server authentication or, on the right, you can see that Active Directory tries to sign you in automatically. I'm going to minimize this left navigation pane so that we have more room for the query editor. Observe that the Active Directory single sign on results in an error because cloud.user @ loonycorn.com is not an Active Directory user. When we had set up the Azure SQL Database, we had specify loonyuser as a SQL Server authenticated user. Type in the password for this user, click on OK to sign in. Off to the left of our query editor is a place where you can view the tables, views, and stored procedures associated with this database. In the center pane here is where you type in your query. This is a brand new database. There are no tables in here yet, so you could try to expand the tables node. You'll find nothing in there. If you try to expand the

views node, you'll find a view for the system database firewall rules. This is a default view of the firewall rules configured on the Azure cloud to access this particular database. We don't really need to worry about this. Let's type in a SQL query here within this query editor to create a new table named mobiles. This has just two columns, ID and MobileName. All queries can be executed by clicking on this Run button available here to the top left of the query editor. Execute this query and a new table will be created. Click on the Results option here. There are no results. Hit the refresh icon on this object explorer off to the left, and you'll see that we now have a table in this particular database. Expand this node and you'll find the mobiles table has been created successfully. Let's insert a few records into this mobiles table by using the INSERT INTO SQL command. INSERT INTO mobiles, ID and MobileName are the columns, and the values that we want to insert are 2300 and iPhone X. Click on the Run button here to execute this query and you'll find that there was 1 row inserted. The query was successful and it affected 1 row. I'm going to run a few additional INSERT commands here. This time I'm going to insert the mobile phone Pixel 3, hit Run, and go ahead and insert the OnePlus 6T as well and hit Run. Now we have 3 records in our table. SELECT * FROM mobiles and you'll find that the three records are now visible in your Azure SQL Database. The query editor allows you to run a number of different queries right here within your browser. Here is how you'd update a particular record in your database using the UPDATE query. Here I update the mobile with ID 2300 to set its name to be iPhone XR. Once you've run the UPDATE command, you can run a simple SELECT * FROM mobiles and you'll see that your record has been updated. Twenty-three hundred is now the iPhone XR. You can also use the query editor to run DELETE queries on your database. DELETE FROM mobiles WHERE MobileName is equal to Pixel 3. This affected exactly 1 row. SELECT * FROM mobiles and you'll see that the Pixel 3 record has disappeared from our database.

SQL Server Management Studio

You can also connect to and work with your Azure SQL Database on the cloud using SQL Server Management Studio on your local machine. In this demo, we'll see how we can set up SSMS on our local Windows machine and use SSMS to load and query data on our Azure SQL Data Warehouse. Let's go ahead and download SSMS first using this link that you see here on screen. This will take us to the SQL Server Management Studio page, and here you'll find a link to download SSMS. SQL Server Management Studio is only available for Windows machines at this point in time. Download this executable file that will allow you to install and set up SSMS. I've stored this executable on my desktop. I'm going to switch over and double-click on this executable to kickstart the install process. Just click on this Install button here and wait for a

minute or so until the install process is complete. Click on Close and you'll now have SSMS set up on your local machine. This little search icon next to my Start button will allow me to search for this particular application. SSMS gives me Microsoft SQL Server Management Studio. Click on it to get it up and running. We want to use SSMS to connect to our Azure SQL Database on the cloud. So our server type is a database engine. The next thing we need to do is to specify a server name. If you remember, when we set up our Azure SQL Database, we had specified that its server was loony.database .windows .net. Before we specify the name of the server, let's head over to the authentication options. We have set up SQL Server Authentication for our Azure SQL Database. We have configured an admin username and password for that when we set this up earlier. In order to be able to connect to our cloud database from our local machine, we need to configure a few firewall rules. Switch over to loonycorndb on our Azure portal and click on the Set server firewall option. This will bring up a web page, allowing you to configure your firewall settings. The name of this firewall rule is ssms because ssms is what we want to choose to connect. Specify the IP address of your local machine. If you don't know what your local IP address is, you can simply visit whatismyip.com, and this will give you your current IP address. You can specify this IP address as your start and end range. If your IP address is part of a subnet in an organization, the subnet range is what you'll specify for the start and end IP. Once you've configured access to your IP address, click on the Save button so that your local IP address has access to the Azure SQL Database that you've created on the Azure platform. Remember that the resources that you create on the cloud are protected and in order to enable access to those resources, such as this database that we have on the cloud, you need to explicitly set up your firewall rules. On this main page for our Azure SQL Database, I'm now going to copy over the server name, loony.database .windows .net. This what I'll paste in as the server name on SQL Server Management Studio. Now let's go ahead and specify the login and password that has access to this particular database. This is loonyuser and type in the password that you've created for your user and click on the Connect button. Wait for a few seconds and you'll be successfully connected to your Azure SQL Database on the Azure cloud platform from your local machine. If you click on the Databases link on the Object Explorer, this will show you the loonycorndb that we had created earlier on the Azure cloud. And if you expand this object, you'll find the mobiles table that we had created earlier using the query editor within our browser window. This is the exact same view that we had within our query editor. Expand the Columns option here and you'll find the ID and the MobileName column within this table. Click on the New Query button, off to the top here, in order to bring up a query editor within SSMS. Here is where you'll type in the queries that you want to execute. You'll need to specify the database that you want this query to run against. By default, a master database is selected. Switch this over to loonycorndb. You want to run the query against the

mobiles table in loonycorndb. Type in a simple query, SELECT * FROM mobiles in our query editor on SSMS, click on the Execute button to execute this query. This query will run on our Azure SQL Database on the Azure cloud platform and here are the results.

Loading and Querying CSV Data

On my desktop, I have the csv file mobile_data_cleaned.csv. This is the data that I'm now going to upload to my Azure SQL Database using SSMS. Select the loonycorndb node on the Object Explorer, right-click, and under Tasks you'll find an option to Import Flat File. Select this option and this will bring up a dialog that allows you to import data into your SQL Database. Click on the Next button here and let's specify the file that you want imported. The Browse button will bring up the explorer window, allowing you to locate this file on your local machine. Here is mobile_data_cleaned.csv. Open up this file. This is a file that we have worked with and are familiar with. Specify the name of the table where you want to import the contents of this file. This is the mobile_data_csv table. Clicking on Next will take you to a pane where you can preview your data. This data looks good. This is the csv data that we want to load into our table. Click on Next once again. Here is where you have the option to modify the data types associated with each of your columns. I'm going to specify that the mobile id is the primary key for this particular table, and I'm going to change the mobile_wt column so that instead of tinyint, it is an int. If there are other changes that you want to make, you can do so on this page. Click on Next here so you can quickly see a summary of what you're going to import into your table. Things look good to me. I'm going to go ahead and hit the Finish button and wait for all of this data to be imported into my cloud database. You've successfully created a new table in your Azure SQL Database and imported data from your local machine using SQL Server Management Studio. Let's query this new table that we've just created. SELECT * FROM mobile_data_csv, and here in the results you can see all of the data has been successfully imported. Once your data is in the SQL Database, you can apply complex SQL queries to extract insights and explore your data. Let's select the ID and PRICE_RANGE from mobile_data_csv, where the phones do not have a DUAL_SIM. Execute this query and you'll find the results here in the bottom pane of your screen. Let's try another query here. I want to find all records where the mobile weight contains the numbers 16, mobile_wt LIKE %16%. You can see from the results displayed here on screen that all mobiles matching this regular expression that we had specified are displayed in our results. We'll now try a slightly different SELECT query. Select certain columns from our database, ID, ram, battery_power, and price_range, where battery_power greater than 1000, and we want the results ordered by ram. And here are the results with the subset of columns that we had

specified, arranged in ascending order of ram. You can run aggregation queries as well. Let's see how many mobile phones are available in each price range in our dataset. The result of this aggregation query will show you that there are around 500 records for each of these price ranges. If you have a query that you use really often while working with your data, you might want to save that query as a view. Click on the Views node here, there are no views created at this point in time. Select this node, right-click, and let's create a new view. A view in a SQL Database can be thought of as a virtual table defined by a query. And that query is what we are about to define here. This brings up a dialog. We want this query to run over the mobile_data_csv table, so that's the table that we select. Close this dialog and let's select the columns that we want to include in this view. Use the checkboxes available here within this little editor window to select the columns that you're interested in. The first column that I've chosen here is the battery_power column. You can use the scroll bar to scroll up and scroll down and view all of the columns in your dataset. Select the mobile_wt column as well. Select the price_range column and observe here below that SQL Server Management Studio has constructed a query corresponding to the columns that you selected from your database table. You can use the File menu and save this query corresponding to your view. Go ahead and save this view, call it the battery_weight_view. All of the views that you create and save will appear under the Views node in your Object Explorer. Once you have a view created, you can query this view exactly like how we would query a SQL table. SELECT * FROM battery_weight_view will list all of the records in that view.

Uploading Data to Blob Storage and Creating a Data Factory

In this demo, we'll load and query JSON data into our Azure SQL Database. This is the same mobile dataset that I've been working with so far, which I've converted to a JSON format and stored in the file mobile_data_cleaned.json. Observe that this file contains a list of JSON objects as specified by the opening square bracket and every object here corresponds to one mobile. The features corresponding to each of these mobiles is represented as a JSON object. Now you can load this data into your Azure SQL Database using SQL Server Management Studio, but it's possible that your data file is present, not on the local machine, but on the cloud somewhere, for example, on blob store on the Azure Cloud platform. Here we are in the Microsoft Azure portal. Let's go to Storage accounts and create a blob storage container. This view shows me the storage accounts that I have created across all locations on the Azure cloud platform. I'm going to click on this dropdown, deselect the Select all option, and select only the storage accounts in US East 2. You'll find that there are no storage accounts here, so I'm going to go ahead and create

one by clicking on this Add button here. This button will allow you to create a new storage account. Storage account is a Microsoft managed service providing cloud storage that is available, secure, and reliable. I have a pay-as-you-go subscription on Azure, so that's what I select here. And I'm going to select the resource group that I had created earlier, loonycorn-rg. I want the storage account to be created in the US East 2 region, that's the region where we've located all of our other resources as well. And I'll give the storage account a meaningful name. I'll simply call it loonystorage. Accept the default values for all other settings and click on the Next button here where you can configure advanced settings if you choose to. All of the transfers to your storage account are encrypted by default, except the default options, and click on the Next button and you'll come to a page where you can configure tags for your storage account. Tags are simply name value pairs that allow you to categorize and identify your resources. We don't need any tags. Click on Next. Let's review the settings for our storage account and click on the Create button here to create this account. Once the storage account has been successfully created, you can click through to this resource where you can set up different kinds of storage. I want to upload my JSON file onto blob storage, so I choose to create a blob storage container. There are no containers in my storage account. I'm going to create a brand-new container by clicking on plus container. This container is called datablob. Click on OK and this container will be created. Click through to the container and here is where I'm going to upload my JSON data. Click on the Upload button here to bring up the dialog, allowing you to upload from your local machine. Click on Browse. Select the file mobile_data_cleaned.json. This is the JSON file that I want to upload onto this blob container. Once this JSON file is present within my blob container, I can now load the contents of this file onto my Azure SQL Database. Use the search bar available on top to search for the SQL databases service. Select SQL databases and select the loonycorndb. This is the database that we have created earlier. I'll now use the query editor to create a new table that will hold the contents of the data that I load in from my JSON file. Authenticate yourself to the query editor using SQL Server authentication and let's write and execute a new CREATE TABLE query. Now notice that this table has just four columns, id, battery_power, bluetooth, and clock_speed. This table will hold just a subset of data from our JSON file. Click on the Run button to execute this query and create this table. Click on the Refresh icon in your Object Explorer. This will update the tables that exist in your database. Click on the Tables node and if you expand it, you'll find the new mobile_data_json table has been successfully created. Now, in order to load the data that is present in a JSON file on our blob storage container, I'm going to use the Azure Data Factory. This is a cloud integration service that allows companies to integrate data from different sources. You might have your data stored on an on-premises SQL Server or on another cloud platform or in blob storage on the Azure cloud. The

Azure Data Factory allows you to bring together data from these disparate data sources onto your SQL Database or Data Warehouse. This will take us to the Data factories page where I'm going to create a new data factory to connect to my blob storage container and bring in my mobile data onto my Azure SQL Database. Let's give a name to this data factory. I'll call this loonycorn-df. This is the pay-as-you-go subscription that I have on the Azure cloud. And I'm going to use an existing resource group. This is the loonycorn-rg that we had created earlier. I'm using the V2 version of this data factory, that is the latest version at the time of this recording, and this data factory is located in US East 2. That's where all of my other resources are located as well. Click on the Create button and then wait for a few seconds for this data factory to be created and deployed. After successful deployment, you can click on the Go to resource button here and here is our Azure Data Factory that we use to load in data from our blob storage container.

Load Data Using Azure Data Factory

Here we are on the Azure Data Factory that we've just created, loonycorn-df. Click on the Author & Monitor option here that will allow us to create a new pipeline to process our data. This will take you to a page which allows you to configure the pipeline of transformations that you want to apply to your data. We want to copy our data from blob storage onto the Azure SQL Database. Choose the Copy Data option, and this will bring up a page, allowing you to configure this pipeline. Let's give our copy data task a meaningful name, copyfromblobtosqldatabase. Click on Next, and here you have the option to specify the source of your data transfer. Now the original data can be in a number of different sources. Let's create a new connection and let's connect to Azure Blob Storage. On the Azure Data Factory, you need to configure your data source and destinations in the form of linked services to this data factory. The options here in this dialog should give you an idea of the disparate sources that the Azure Data Factory supports. I'll now configure this linked service to our blob storage container. I'll give it a name, AzureBlobStorage1. We'll use the AutoResolveintegrationRuntime, which is the default for all cloud resources. This is the integration infrastructure that Azure Data Factory uses to connect to your source. Let's scroll down here on this page and let's specify our Azure subscription. It is a pay-as-you-go subscription, as we've discussed earlier. We also need to configure the storage account for our blob storage container. This is the loonystorage account. These are the only details that we need to configure. For the source of data pipeline, click on the Finish button here, and we've set up a new Azure blob storage link service. I'll now move on to specify the input file from where I want to copy data. Click on the Browse button here, and within the datablob container that we had set up on our blob storage container, select the mobile_data_cleaned.json. Since this is a file and not

a folder, you can uncheck copy file recursively, and then click on the Next button. The data factory has auto detected that this file is in the JSON format. Now, you can specify the pattern of your file. Whether it's a set of objects or a list of objects. Our file is an array or list of objects and that's the option that I've chosen here. Once you have the right file format and file pattern specification, you should be able to view a preview of your data right here on the screen. If you scroll down in the file format settings pane, you should be able to see how the individual columns of your data have been extracted from the JSON objects in your file. I'll now take a look at the schema specification for my data using the Schema tab here at the bottom, and here you can see how the column names map to data types. Now if you remember, our table in the Azure SQL Database contains only a subset of columns from our original dataset. You can use the schema option here in order to select those columns that you're interested in. We want the data in the columns id, battery_power, scroll down, choose bluetooth and clock_speed as well. With our source column selected, we can click on Next and move on to configuring the destination of this data pipeline. Click on the Create new connection button to configure our destination. Our destination is the Azure SQL Database. I search for Azure SQL and select the Azure SQL Database here in my list of options. Let's now configure our destination link service. This is the AzureSqlDatabase1 service. Using the AutoResolveintegrationRuntime as before, let's scroll down and specify the pay-as-you-go subscription that I have on the Azure cloud. Next we specify the server where we want to load this data. This is the loony server that we had set up earlier, remember the full name is loony.windows .database .net. Use the dropdown next to the database name and select the loonycorndb. This is the database where we have the mobile data JSON table. We'll connect to this database using SQL Authentication. Remember, we had set up loonyuser earlier. Specify loonyuser as the user name. Specify the password for this user and click on Finish. This will create the destination link to the Azure SQL Database. Select the SQL database link and click on Next where we can now specify the table into which we want to load this data. This is the mobile_data_json table. Clicking on Next here will take you to a screen where you can configure the column mappings for your table. You can specify what columns in your source data should be mapped to which columns in your destination database. You can double check that the column mappings here have been set up correctly and click on the Next button to continue. On the Settings page here, you can configure properties for your copy data operation. The fault tolerance setting that I choose is to abort activity on first incompatible row. Here is where you can set up a staging environment and configure other advanced settings as well, such as the parallelism you want for your copy. I'm going to choose the default values and click on Next. This will take you to a page that you can use to review your copy data pipeline from Azure Blob Storage, that is the source, to an Azure SQL Database on the cloud. You can scroll down and

review all of the other configuration that we specified for the source, as well as the destination. Observe that the timeout for this copy is 7 hours by default. Click on Next here and this will kickstart your data factory pipeline to load in data from the blob storage onto your Azure SQL Database. With pipeline deployment complete, let's monitor this by clicking on the Monitor button here. A pipeline is currently in progress. The copying is currently going on. Click on Refresh here and you'll find that in a minute or so, the pipeline will have succeeded. Let's confirm whether data has actually been loaded into our SQL Database by switching over to the other tab. Search for SQL databases and click on this option here to head over to the SQL database that we'll be working with, this is the loonycorndb. Click through and bring up the query editor so that we can query data to check whether the contents have been loaded successfully using the data factory. Authenticate ourselves before using the query editor, and let's run a SELECT * FROM mobile_data_json. Execute this query and you'll find that our data has been successfully loaded onto this table.

Summary

And with this demo, we come to the very end of this module on loading and processing data using relational databases. Relational databases have been around for a really long time. In this module, we got a taste of how it is to work with the Azure SQL Database, the managed service that Microsoft offers on the Azure cloud platform. We saw that Azure supports a number of deployment options for its SQL Database, single database, elastic pool, and a managed instance. We chose a single database and we queried data that we loaded into this database using the query editor available on the Azure portal. We also saw that it was possible to connect to our SQL Database on the cloud from our local machine by installing and using the SQL Server Management Studio, or SSMS. We use SSMS to load the contents of a relational database onto the cloud and query that data. We also worked with Azure Data Factory pipelines. These are extract, transform, load pipelines that bring in data from disparate sources. We used Azure Data Factory to load JSON data stored in a blob storage container into a SQL Database and then queried that data. Now that we've collected all of our data together on our Azure SQL Database, in the next module we'll see how we can represent insights obtained from data using the Azure Data Studio, as well as Power BI.

Representing Insights Obtained from Data

Module Overview

Hi, and welcome to this module on Representing Insights Obtained from Data. We've collected our data, brought it together into a single source, and we've prepared our data. In this module, we'll talk about the different kind of numeric data that you can use in analysis. We'll talk about and see some techniques to plot continuous data. We'll understand what categorical data means and see a few examples of representing categorical data. We'll talk about other kinds of data that you might work with for analysis, such as text and image data, and we'll discuss what representations of such data make sense before they can be used in ML models. On our hands-on demos, we'll work with two specific tools. We'll use Azure Data Studio to query and model our data, and we'll use Power BI for visualization. We'll see that Azure Data Studio is a cross-platform tool that works on Windows, Linux, as well as Mac machines. Whereas Power BI, at this point in time, is available only for Windows, but offers very powerful drag and drop visualization.

Continuous and Categorical Data

All of the data that you work with for analysis to build data models, machine learning models, is numeric in nature. The kind of data that is used in analysis is typically divided into two broad categories. The first of these is continuous data. Continuous data can take on a continuous series of values within a range, such as house prices, stock prices, the temperatures recorded over a year, the height of individuals, all of these are examples of continuous data. The other kind of data that you work with for analysis is categorical data. Categorical data is made up of discrete values. The value assigned to your categorical variable can only be one of a discrete and finite set. True or false? Zero or one? Days of the week. Months of the year. These are examples of categorical variables. Both continuous, as well as categorical data, have to be represented in numeric form. Continuous data is usually numeric in nature, whereas categorical data may be expressed in form of strings or categories. This categorical data has to be converted to numeric form before it can be used in your models. Whatever features you're working with, whether it's just ordinary numeric features or text and image data, all other forms of data must be converted to either continuous values or categorical values before you can use them for analysis. And remember, both continuous, as well as categorical values, have to be expressed as numbers. Now that we've

understood what continuous and categorical variables mean, let's compare and contrast the two. Examples of continuous variables are the height or weight of individuals. They can pick on any value within the range. Examples of categorical or discrete values are day of the week, month of the year, the gender, male or female, or a letter grade that you might get for your assignments. It's pretty obvious, as their name suggests, continuous variables can pick on any value, whereas with categorical variables there are a finite set of permissible or discrete values possible. When you're working with machine learning, regression models are what you build to predict the value of a continuous variable. Classification models is what you'll build when you're predicted output is a category. Another significant difference between continuous and categorical variables is the fact that continuous values can always be sorted based on magnitude. Their magnitudes are significant. A house costing $100, 000 is always cheaper than a house costing $200, 000. Categories, on the other hand, may or may not be sorted. Male, female, really no sorting order there. Grade of A, B, and C, yes that's sortable. Let's quickly discuss the four types of categorical data that you might work with. Binary categories are where there are just two permissible values, yes or no, male or female. Multiclass categorical variables have multiple permissible values, digits from 0 through 9, days of the week. Nominal categorical data is inherently unordered, male or female, days of the week. There is no specific ordering here. Categorical data that is ordinal in nature may have a possible ordering. If you're taking about letter grades, an A is definitely better than a B.

Numeric Representations of Text Data

Text data is, well, text. But data analysis and machine learning models require your data to be numeric. In this clip we'll talk about how we can convert text data to numeric form. Let's consider a document, which is essentially just a restaurant review. This is the text that makes up this document. Model this document as an ordered sequence of words. This will involve splitting this document into words, or tokenizing this document. All of the individual words of this document are extracted and set up in an ordered form. In order to convert this text document to numeric form, we'll now have some kind of numeric encoding for each word in this document. Let's say the word this is W0, it's numeric encoding is X0. Then we'll have an encoding X1 for the next word, and an encoding X2 for the next word, and an encoding Xn for the nth word in this document. Once all of the individual words in this document have numeric form, this document can be represented as a tensor or a matrix of numbers. Well, this sounds relatively straightforward, but the interesting detail here is how do we convert every word to a numeric representation? As you might imagine, there are several techniques that you can use for this. Here

are three broad categories. You can use one-hot encoding, frequency-based encoding, or prediction-based encoding for your words. I'll now give you a quick overview of how each of these encoding types can represent text in a numeric form, starting with one-hot encoding. One-hot encoding sets up a vector to represent every word. The size of this vector is equal to the number of words in the vocabulary of your corpus. And every word in the text is represented by the presence or absence of the word. A number 1 for a particular word indicates that a particular word is present in a document, 0 indicates that that word is absent. This is the simplest way to convert text to numeric format and is widely used. Let's now move on to frequency-based encoding. One-hot encoding does not tell you how often a particular word occurs in text. Frequency-based encoding does. There are three categories of frequency-based encodings. Count vector encodings, which give you a count of how often a word occurs in a document. Then there is TF-IDF encoding, where TF stands for term frequency and IDF stands for inverse document frequency. And finally, you can encode words based on co-occurrences. Co-occurrences capture how often a particular word occurs in the context of its surrounding words. Frequency-based embeddings using count frequency is just a little bit of an improvement over one-hot encoding. This also captures how often a particular word occurs in a document. Instead of using 1 and 0 to represent the presence or absence of a word, we'll say 2 if a word occurs twice in a document or 3 if it occurs thrice. These are the counts, or frequencies. TF-IDF is a very popular representation that tries to capture how significant a particular word is in a document and across the corpus. The TF-IDF is a combination of term frequency and inverse document frequency. The term frequency tries to capture how often a word occurs within a document, and the inverse document frequency tries to capture how often that word occurs across the entire corpus. Here is how TF-IDF generates numeric representations for words. If a word occurs more frequently in a single document, that word might be important and it's up weighed. However, if a particular word occurs very frequently across the entire corpus, that is probably a common word, like a, an, or the. In that case, that word is down weighed. You can also generate numeric embeddings for words based on how often a particular word co-occurs with other words that surround it. This is based on the idea that similar words will occur together and will have similar context. Generating co-occurrence representations involves the use of a context window. This is a window centered around a particular word and it might include a certain number of neighboring words, to the left, to the right, or both. It's possible to build up a co-occurrence matrix representation of the word using this context window. Co-occurrence is basically the number of times two words, w1 and w2, have occurred together in a context window. Of all of the possible ways to numerically represent a word, prediction-based embeddings are the most powerful. Prediction-based embeddings try to capture not only the meaning of a particular word, but also

the semantic relationships that that word might have with other neighboring words. Generating prediction-based word embeddings involve the use of machine learning models, such as neural networks. Popular word representations generated using this technique are Word2Vec and GloVe.

Representing Image Data as Matrices

In this clip, we'll see how we can represent image data in numeric form so that it can be used in data analysis and to build ML models. When you work with images, you're just working with numeric matrices. An image under the hood is just a matrix. And every element of this matrix represents information for a single pixel of this image. Imagine that pixel information for an image is stored within this grid that you see here on screen. How is pixel information represented? Well it depends whether it's a colored image or a grayscale image. Colored images are also referred to as RGB images, red, green, and blue. Every pixel contains RGB values, which represents the color for that particular pixel. RGB values are just integers from 0 to 255. A value for R 255 is a red colored pixel. As you can see here, the values for G and B are 0. If G is equal to 255 and the values for R and B are 0, that's a green colored pixel. And finally, when B has a value of 255 and R and G are 0, that's a blue colored pixel. There are three values that you need to represent color. This is a 3 channel image. Pixels within a grayscale image need just one value to represent information for that pixel. Each pixel has just intensity information and a pixel value is typically from 0 to 1. So you might say that the intensity of a particular pixel is 0.5 or 0.75. There's just one value to represent intensity, this is a single channel, or one channel image. Now that you've understood how pixel information is represented, here is an image. Images have height and width, which means that images can be represented using a 3-dimensional matrix. The first dimension is the height, next is the width, and the third is the number of channels. Here on the screen, we have two images that are 6 pixels by 6 pixels. So the matrix shape will be 6, 6, 1 for the grayscale image. Remember we have just a single channel to represent intensity information in grayscale, and the shape of the image matrix for a colored or RGB image will be 6, 6, 3. The last dimension is 3, 3 channels to represent RGB information. Typically if you're performing data analysis, especially machine learning, you won't work with a single image at a time, you'll work with a list of images. Provided all of the images are of the same size, you can represent this as a four dimensional matrix. The shape of the dimension at the very right represents the number of channels in each image, which will be three for our gp images. The dimensions at the center represent the height and width of each image in the list. Remember this 4-D representation works only if each image in the list has the same height and width. And finally, the shape of the very first dimension in this 4-D matrix is the number of images. It happens to be 10 here, but it can be anything.

Azure Data Products

We've already created and set up an Azure SQL Database on the Microsoft Azure platform and interacted with it using the SQL Server Management Studio, or SSMS. Based on your use case, Microsoft offers a host of products that you can use to interact with the Azure SQL Database. We are going to work with the Azure Data Studio here in this module. We've already got a taste of working with SQL Server Management Studio, which you install on your local machine, and you can use this tool to connect to SQL Databases on your local machine or on the cloud. SSMS is an extremely popular integrated environment for all SQL services. We can use it with SQL Server, Azure SQL Database, and the SQL Data Warehouse on the Azure cloud platform. Another option that you have is to use the Azure Data Studio. This is an integrated environment for querying and visualizing data on Azure, as well as for on-premises SQL Servers. Basic operations that you might want to perform are common between the two, but Azure Data Studio is designed for data professionals rather than database administrators. Let's quickly compare and contrast SSMS versus Azure Data Studio. SSMS is meant for database professionals, those who are focused on database administration configuration and management. Azure Data Studio, on the other hand, is meant for data professionals, those who are looking to extract and analyze data stored in a database. The focus of SSMS is on database management, whereas for Azure Data Studio, the focus is on querying and visualization of data. SSMS has extensive wizards, which you'll need to use when you're configuring your database, whereas with Azure Data Studio there are not as many wizards for you to work with. At the time of this recording, SSMS is available only for Windows users, not for Mac or Linux machines, whereas Azure Data Studio is available for Windows, Mac, as well as Linux operating systems. When you're working with SSMS, there is little emphasis on using the command line. Whereas with Azure Data Studio, it's meant for power-users of sqlcmd or PowerShell. In this module, in addition to Azure Data Studio, we'll also visualize our data using Power BI. This is a business analytics app with powerful drag and drop visualization and data exploration capabilities. Power BI is powerful, simple and easy to use, and closely integrated with Microsoft and Azure data services.

Installing and Working with Azure Data Studio

At this point, we've collected all of the data that we want to work on in an Azure SQL Database on the cloud. In this demo, we'll see how we can use Azure Data Studio to query data and represent insights. Azure Data Studio is a free cross-platform tool that Microsoft offers to connect and work with SQL databases, which are present on-premise or on the cloud. Unlike SQL Server Management Studio, which works only on Windows machines, Azure Data Studio can work on

Mac, as well as Linux machines. I'm currently on a Windows machine, so I'm going to scroll down and use this installer here in order to install Azure Data Studio for Windows. I'll save the installer executable onto my desktop and I'm going to switch over to my desktop and double-click on this installer to kickstart the install process. Click on the Next button here and accept the terms of the agreement to set up Azure Data Studio. Specify the location on your local machine where you want this program to be installed and click on Next once again. The name of this program is Azure Data Studio. Click on Next and move on. I want this to be added to my PATH environmental variable, select that option, click on Next, and you can now click on the Install button to start the install process. Wait about a minute or so for the install to be complete and click on Finish. Azure Data Studio will start up and bring up the welcome screen where you can specify your connection. We want to connect to Microsoft SQL Server, that is our Azure SQL Database on our Azure cloud platform, and we want to use SQL authentication to connect to this database. Choose SQL login here. The name of our server is loony.database .windows .net. This server name is available on your Azure SQL database page for loonycorndb. Specify the username and password for your SQL login, that is loonyuser and whatever password that you created. I'm going to select the checkbox to remember password. I'll stick with the default values for all other options and click on Connect. We'll now successfully connect it to our Azure SQL Database on the cloud using Azure Data Studio on our local machine. On the Connections pane here on the left, if you expand the Databases node, we'll find our loonycorndb listed under there. If you expand the loonycorn database, under Tables you'll find the three tables that we've created earlier, mobile_data_csv, mobile_data_json, and mobiles. We had also created a view using SQL Server Management Studio and that you'll find listed under the Views node as well. This is the battery_weight_view. If you close the server dashboard view that shows up by default, this will automatically bring up the query editor that will allow you to type in your query. If we want our queries to run against loonycorndb, let's write our first query here to create a new test table, test_mobile_data. Click on the Run button here to execute this query. As you can see, the interface on Azure Data Studio is very similar to the query editor that we saw on the cloud, as well as the SQL Server Management Studio. This makes it very easy for a user to transition between these two. Let's write an insert command here to insert some data into this test_mobile_data table. This is a mobile with id 212 and name BlackBerry Key2. Run this command and one row will be inserted into this table. Let's execute another simple query here, SELECT * FROM test_mobile_data. Click on Run and here is the one record that's present in this table. The Azure Data Studio is more than just a simple query editor. It offers intelligent SQL snippets as well. Type in SQL and this will bring up several template options that you can then choose from. I'm going to select the sqlDropTable template, and you can see that a template in

SQLQuery has now been populated in my editor. You can now edit this template to drop the table that you're interested in. I'm going to change the table name to be test_mobile_data and drop this table. This saved us the trouble from dropping the actual query. We simply used the template and filled in the specific details for our particular query. Click on Run and this table will be dropped. Now if you try to execute the query, SELECT * FROM test_mobile_data, you'll find that this query results in an error as test_mobile_data no longer exists.

Visualizing Insights Using Azure Data Studio

In this clip, we'll continue working with Azure Data Studio. I'm going to run an ALTER DATABASE query to turn on Query Store. Query Store on the Azure SQL Database captures a history of queries, plans, and other runtime statistics to help us with performance monitoring. Once we turn Query Store on, we can run a number of different queries and view the performance of these queries on our dashboard. Click on this little gear icon here off to the bottom left. This will bring up a menu and select Settings from that menu. This will take you to a page where you can tweak the configuration settings associated with your Azure Data Studio. On the left, you can see the various categories of settings that are available. If you're looking for a specific setting, you can use the search box available on the top. I'm looking for dashboard settings. And I'm going to select this option to edit these configuration settings in settings.json. This will allow me to configure widgets that I want on my monitoring dashboard. Settings.json contains a bunch of different information, all specified in the JSON format. You can see there is JSON for the connection groups that have been created and the different connections that exist to your database. We have just the one connection here to loony.database .windows .net. That is our server name. I'm going to paste in some JSON here to configure a few additional widgets on my monitoring dashboard. The JSON key for the monitoring widgets is dashboard.database .widgets, and within that we have JSON objects specifying the different widgets that we want to view. I've configured a slow queries widget, a Tasks widget, and an explorer widget. Observe that every widget has a name, size, and a widget type. All of these are built in widgets available on the Azure Data Studio. Hit Ctrl+S to save the settings in settings.json and select the loonycorndb, right-click and choose the Manage option. And this will bring up the monitoring dashboard for this particular database and you'll find here the widgets that you configured. Here is our slow queries widget with tracks, the time taken for every query that you executed on your database. The visualization that you see here is interactive. You can hover over to view additional details or you can use this three dot menu available here on top to view additional details about each of these queries. Click on Show Details and this will bring up a tabular representation of all of the

queries and the time taken. Every query is associated with an id. You can select a particular query. I've selected the one with id 121, and view additional details of that query in the item details table here at the bottom. You can see that this was an insert query and see when it was last executed and how long it took. Close this pane and let's head back to the settings.json file that is available here on this tab. Let's configure an additional widget here that will tell us how much space is occupied by each of our tables. This is the table-space-db-insight widget. Now let's go back to our dashboard view, select the loonycorndb, right-click, and select Manage. Here is the same monitoring dashboard that we saw earlier. It has an additional widget, which indicates how much space is occupied by each of our tables. Let's close this dashboard view and take a look at some of the visualization features offered by Azure Data Studio. Here is our query editor, and I'm going to run an aggregation query here. I'm going to count the number of mobile phones available at each price range in our dataset. I've grouped this data by price range, and I've selected the price_range and COUNT * AS count. Execute this query by clicking on the Run button, and the results will be displayed in a tabular format, as you can see here in this pane. In order to visualization this information using a chart, off to the right here, you'll see this little chart icon, which you can then click. You can then choose the type of chart you want to use to represent this data. The default here is a bar chart. I'm going to switch the data direction to be vertical, and here is the data represented in the form of bars. I feel that this data is best represented in the form of a pie. Change the chart type to be a pie chart and here is a pie chart with our price range and the count of mobiles. In our dataset, you can see that there are almost an equal number of mobiles in each price range. I'll now use the query editor to run and visualize a slightly different query. I would select the talk time and the average batter power across all of my mobiles where I group and order by the talk time. Run this query, the results are displayed in the table format by default. Click on the chart icon to bring up the visualization menu. Azure Data Studio has picked up our last representation, the pie chart. This may not be the best option here. Use the dropdown to select another chart type. I'm going to go with the table representation of this data. Here is a table with two columns, talk time, and the average battery power for all of the mobiles with the same talk time. Let's run another application query. I'm going to select the price range and the average ram for each price category from our mobile data set. Execute this query. This is displayed in the form of a table, that is the last representation that we'll pick. Let's switch the chart type over to a bar chart so that we can visualize this information in the form of vertical bars. You can see that the average ram for more expensive mobiles, that is in category 3, tends to be higher as compared with lower priced mobiles. I'm now going to hit Ctrl+S in order to save this chart visualization. I'm going to call the file custom_bar_chart. The tab associated with this query and chart has now been updated. It's now called custom_bar_chart.sql. Observe that your chart

configuration in here has this option to use column names as labels for your chart. Check this and you'll find that the title of this chart is now updated to say average ram. If you now click on this Create insight button available here, this will allow you to set up this particular chart as a widget, which you can then display on your monitoring dashboard. Here is the JSON representation of this widget. You can copy this JSON over, select the gear icon here at the bottom left, and choose Settings from the popup menu. In the settings page here, search for dashboard and select the dashboard widgets option and edit in settings.json. This is a JSON file that we are familiar with based in the JSON that we just copied over for our newly created custom widget. Hit Ctrl+S to save the settings.json file, select your database, right-click, and choose the Manage option in order to view your dashboard. Select the loonycorndb. That is the dashboard that we want to see. And here at the bottom is our custom visualization. The average ram for each price category displayed as a bar chart.

Installing and Visualizing Data in Power BI

In this demo, we'll use the Power BI tool that Microsoft offers in order to visualize the data that was stored on the Azure SQL Database on the cloud. If you've worked with Microsoft tools, it's quite possible that you've worked with Power BI. It's a widely used business analytics and visualization tool, which can integrate with your cloud database. Here in this demo, I'm going to download Power BI for my Windows desktop and use Power BI to connect to Azure SQL Database on the Azure cloud. Click on the Download button here on this page, Power BI for the desktop is absolutely free for you to use. Open up the Microsoft store and this will take you to a page where you can download the installer. Click on Install and wait for a couple of seconds until the product is installed on your local machine. Click on the Launch button here to bring up the Power BI desktop. Let's close this dialog asking us to sign in. We'll just work with Power BI on our local machine. Before we create visualizations using Power BI, we first need to set up a connection to the data that we want to work with. Click on the Get Data option here. Here are all of the data sources that you can use with Power BI. Click on the More button here at the bottom and this will show you the complete list of sources that can be integrated with Power BI. Select the Azure category here. Our database is after all located on the Azure cloud platform. And select the Azure SQL Database option and click on Connect. Enter the connection details into the dialog that pops up. The server is loony.database .windows .net. The database that we want to connect to can be provided here or you can leave it blank. And the data connectivity mode that I have chosen here is import. This will import the data from an Azure SQL Database onto my local machine. You can also choose to use the DirectQuery option if you want to. Click on OK here, and

in the next screen, choose the authentication mode that you'll use to connect to your database. I'm going to use the SQL authentication mode, that is the database option, loonyuser is the name, and specify the password. Click on Connect and the connection will be established to your Azure database where the database that we want to work with is the loonycorndb. Select the loonycorndb and select the table that you want to query. This is the mobile_data_csv. Here is a preview of the data that I'm going to import into Power BI on my local machine. Select mobile_data_csv and load this data into Power BI. All of the fields and values in the mobile_data_csv table is now available within Power BI, and you can access these fields using this pane off to the right. Power BI allows you to build up visualizations quickly and intuitively using simple drag and drop methods. Select the kind of chart that you want to visualize your data. I've selected a pie chart here. Now I'll use these field values on the right to select what I want to display in my pie chart. Select the price_range category. This will form the values in my pie chart. By default, fields are automatically added to values. I want the price_range to be the legend though. So I'm going to drag price_range and move it up to the legend section on my pie chart visualization. The values that I want to display in this pie chart is a count of the number of mobiles that I have from each price range. Click on the price_range field and the values, this will bring up a menu, which will allow you to specify the aggregation that you want to perform on the price_range. I want to count the number of records for each price range. And here is the resulting pie chart. You can see that there are roughly equal number of mobiles in every price range in our dataset. Power BI gives you very granular options to customize your visualization. Click on this customization icon here and let's change the colors that you use to represent your data. Click on the data colors option and here you can configure custom colors for each section of your pie. I'm going to change section 2 to be a bright pink in color. And you can see that this particular section is now a bright pink/purple in color.

Creating Different Visualizations in Power BI

In this clip, we'll explore some more visualizations that we can create using Power BI. Select the line chart option on Power BI. A line chart is created within your dashboard pane. You can always select this chart on your main edit pane and move it to the location where you want it to be. This is a line chart that I'll use to visualize the camera megapixel values. Select the pc_megapixel values for the primary cameras and have that be the axis value for your line chart. Simply drag the pc_megapixel field and drop it onto the axis section of your pane. I'll now select the megapixel values for my front camera fc_megapixel. Select this field and drag it onto the Values section of your pane. This upwards sloping line tells us that mobile phones that tend to have good primary

cameras tend to have good front cameras, or secondary cameras, as well. I don't want the values to represent raw fc_megapixel. Select the field and choose the average aggregation. So the chart now displays the average of fc_megapixel for every value of the primary camera megapixel. I'm now going to customize this line chart to have trend lines, click on the search icon here and choose the Trend Line category. Click on the Trend Line option here to add a trend line representation for our line chart. You can see a dotted line is added to our chart showing us the general tread, which is upward. Let's change the transparency of this trend line so that its opacity is reduced. It's more transparent and in a lighter shade. And let's move on to adding another visualization, this time we'll select a bubble chart. Select the bubble chart and it'll be added onto your edit pane. You can resize it as you wish to. Along the X axis of this bubble chart I want to represent the price_range, the fourth category of prices to which mobiles in my dataset belong. I'll drag this field onto the X axis section. Power BI automatically performs a sum aggregation. Bring up the menu and say Don't summarize so that we have individual price ranges along the X axis. Along the Y axis, I'll represent the ram values or the memory available for all of the mobiles in each price range. Select the ram field, it'll be automatically added to the Y axis. Once again, Power BI applies the sum aggregation to the ram data. Select the field and choose Don't summarize so that you view the individual data points in the form of a bubble chart. We already know that more expensive mobiles in price_range category 3 tend to have higher values of ram.

Summary and Further Study

And with this, we come to the very end of this module on representing insights obtained from data. We first spoke about the two categories of numeric features, continuous data and categorical data, and understood the differences between the two. We saw how we could plot and represent both of these types of data using visualizations on the Azure Data Studio, as well as Power BI. Data comes in many forms, but for analysis you need to convert them to a numeric representation, and we saw how we could do that with both text data, as well as image data. In this module, we worked with two more Microsoft products that integrates with the Azure SQL Database, the Azure Data Studio, which is primarily used for querying and visualizing data, and Power BI, which is a very simple intuitive and easy to use drag and drop visualization tool. If you're interested in data analysis and Microsoft Azure products, here are some other courses on Pluralsight that you can watch. Summarizing Data and Deducing Probabilities will discuss univariate, bivariate, and multivariate analysis of data and working with Bayes' rule to solve business problems. Another course that you might be interested in is Experimental Design for Data Analysis. Here we'll talk about the basic principles involved in designing a machine learning

experiment and work with the Azure ML Studio. That's it for me here today. I hope you had fun with this course. Thank you for listening.

Course author

Janani Ravi

Janani has a Masters degree from Stanford and worked for 7+ years at Google. She was one of the original engineers on Google Docs and holds 4 patents for its real-time collaborative editing...

Course info

Level	Beginner
Rating	★★★★½ (23)
My rating	★★★★★
Duration	2h 45m
Released	19 Jun 2019

Share course