Exploratory Data Analysis with R

by Matthew Renze

Start Course

Bookmark Add to Channel Download Course

Table of contents Description **Transcript** Exercise files Discussion Learnin

Introduction to R

Introduction

Hi, my name is Matthew Renze with Pluralsight. In this course you'll learn how to perform Exploratory Data Analysis with the programming language R. When you're finished with this course you'll be able to use exploratory data analysis techniques and R to solve day to day developer tasks, like transforming data files, detecting anomalies, and visualizing patterns and data. So let's get started. As an overview, first we'll start with an introduction to the R programming language. We'll learn what it is, why it has become so popular for data analysis, and how to perform basic programming tasks using R. Next we'll learn how to use R to load, transform, clean, and export data. This is usually the most difficult and time consuming task in any data analysis, but it's a very important step and an extremely useful skill to have. Then we'll learn how to calculate descriptive statistics using R. Descriptive statistics are numerical quantities that provide us with the basic shape and feel of the data they describe. Next, we'll learn how to visualize data using the basic R plotting system. Data visualization is an extremely useful technique for finding patterns and data sets by representing the attributes of the data via visual means. Finally, we'll look at several steps that go beyond R and exploratory data analysis. First we'll cover a few alternatives to using R for performing exploratory data analysis, then we'll look at a few data analysis techniques that can be performed with R that go beyond exploratory data analysis. The only prerequisite for this course is that you have experience with at least one C-like programming language. Languages like C++, C#, Java, JavaScript or Python are all sufficient to understand the concepts in this course. As long as you have a basic understanding of programming constructs, control structures, and data structures, you should do just fine. The intended audience for this course are developers who work with data on a daily basis and want to have the skills necessary to explore and analyze this data quickly and efficiently, data analysts with a bit of programming experience who want to learn how to perform exploratory data analysis using R, or anyone in information technology with basic programming experience and a desire to learn how to transform data into actionable knowledge. Whether you realize it or not, there's a flood of data coming our way. In fact, the flood is already here and it's growing exponentially each year. In the next decade or so, you're going to see two completely different outcomes for people, businesses, and governments based on whether they learn to use these data to their advantage or not. Essentially these people, businesses, and governments are either going to sink in the sea of data or they're going to learn to swim. This is which it's important that we learn how to work with data and transform it into actionable knowledge. We want to be the people that are swimming in the data driven economy, not the ones that are sinking. This same sentiment has been expressed by experts across various industries. In fact, articles like these pop up on a daily basis these days. We now live in an economy where data is extremely inexpensive to produce, store, and process. We essentially have more data than we know what to do with. So the scarce resource in this data driven economy are people with the skills and tools to work with and extract value from data (Loading). So you might be thinking to yourself, I'm not a statistician or a data scientist, so how does this apply to me? Well, as a software developer I often perform log file analysis; analyze software for performance issues, analyze code metrics for code quality, detect anomalies in source data, transform or clean data files to make them usable, and help decision makers make decisions based upon data. It's much easier to extract value from data if you have the skills and the tools necessary to transform, analyze, and visualize data.

Intro to R

The first question you might have is, what is R? R is an open source programming language derived from a statistical programming language called S. S was invented by Bell Labs back in 1976. The S stands for statistics. R is both a language and an environment. That means we have both a programming language, which we can use to write code, and an environment within which we can write that code. R provides methods for both numerical and graphical data analysis. That means we can work with data, both numerically or visualize the data graphically. R is also a cross platform, which means it runs on Windows, Mac, and Unix systems. In addition, R is actively under development with new packages being created every day; it has a large and active user community, it is very modular and extensible, it has over 6700 extension packages at the time this course was created, and best of all, R is free open source software, which means that it is free as in beer, meaning anyone can use the software without cost, and free as in speech, meaning you can view the source code, learn from it, modify it, or redistribute it as you wish. As I mentioned, R is a very popular programming language, so here's a quick visualization from RedMonk to show us where it lies in terms of programming language popularity. On the X axis we have programming language popularity rank on GitHub. On the Y axis we have popularity rank on Stack Overflow. As you can see at the top corner, we have the very popular general purpose programming languages, for example, C#, Java, Python, and Ruby. In this next cluster we have the popular more specialized languages, for example, Matlab, Lisp, and Erlang. And finally, down here we have the less popular programming languages. Here we have R, right at the top of the specialized languages. I should note that R will most likely never be as popular as the general purpose programming languages. It's specialized to work with data extremely well, but you probably won't be using it to build your next ecommerce website anytime soon. This is what R looks like. As you can see, we have both a programming language and an environment. In addition, you can see that R can do both numerical analysis and graphical analysis as well. You can download a free copy of R from the R project website at www. r-project. org. The software installer will walk you through the steps to install R on your computer.

Hello World Demo

For our first code demo, we're going to complete a Hello World example using the R programming language inside of the R development environment. To perform Hello World, we want to set a variable X equal to the character string Hello World, and then print this variable to the console window. First we type into the console x followed by the assignment operator, then Hello World surrounded in double quotes. When we press Enter, the character sting Hello World is stored in a variable named x. Next, we type the print function and pass x in as an argument to the function. When we press Enter, the value that was assigned to the variable x, that is Hello World, will be output to the console window. And there you have it; we've successfully printed Hello World to the console using a variable in R.

RStudio

Rather than working with the out of the box R IDE in this course, we're going to use a more powerful IDE for R called RStudio. It's still R under the hood, but the IDE is much more user friendly and it makes it much easier to do code demos. We'll walk through the IDE during our next code demo, which is coming up next. You can download a free copy of RStudio from the RStudio website at www. rstudio. com. The software installer will walk you through the steps to install RStudio on your computer. There are four main window panes in RStudio. First, in the upper left hand corner, we have the source pane. It contains the scripts that we will be running for each of our demos. Next, in the lower left hand corner we have the console pane. This is where we can interactively program in R. We type commands and output appears in the console window. Then in the upper right hand corner, we have the environment pane. The environment pane shows us the state of the variables that we are currently using in memory. Finally, in the lower right hand corner we have the miscellaneous pane. Here we'll find tabs for navigating the file system, viewing charts and graphs, and reading the help documentation. In order to make the code demos run more smoothly, we're going to be executing scripts from the scripts pane line by line rather than typing everything by hand. We could do everything by hand, like in the Hello World demo, but typing during demos is error prone and doesn't help to move the demos along any faster. It'll work like this, I'll place my cursor on the line of script I plan to execute, then I'll press Ctrl+Enter, which will execute that line of code in the console. You'll see both the command being executed and the result of the command in the console, so you can just keep your eyes on the console window most of the time. Watching the script window will essentially just be a distraction. So see how this will work, we'll just walk through the Hello World demo again in RStudio. First in the console, we'll set x equal to the character string Hello World. Next, in the environment pane we can see that the variable x has been assigned the value Hello World. Then in the console we'll print x, which writes Hello World to the console window. Easy enough? Great. Let's look at the basic language features now (Loading).

Language Basics Demo

As we saw in the Hello World demo, we can assign a value to a variable using the left arrow assignment operator. This operator assigns the value on the right to the variable on the left. In addition to the left arrow assignment operator there are two other value assignment operators in R. The second assignment operator is the equal sign assignment operator. For the most part the equal sign assignment operator behaves similar to the left arrow assignment operator. However, there is a slight technical difference regarding the scope of the variable assignment. This technical difference is outside of the scope in our demos in this course though. The third assignment operator is the right arrow operator. This operator assigns the value to the left to the variable on the right. The right arrow operator is rarely if ever used through. The general preference amongst most experts in the R community is to use the left arrow assignment operator. The primary reasons for this are historical and to avoid potential ambiguity. For all of our demos in this course we will be using the left arrow operator exclusively, however, you are free to use whichever operator you prefer with some very small technical exceptions. Rather than having to type print(x) every time we want to write the value of a variable to the console, R has a feature called implicit printing. Implicit printing allows us to simply type the name of the variable, for example x, and the value of the variable is printed to the console. We will be using implicit printing going forward in all of our demos. To create variables in R we just assign a value to a variable. For example, we can create a logical or Boolean variable and set its value equal to either true or false. We can create an integer variable by setting its value to an integer and using the capital letter L suffix. We can create a numeric variable by setting its value to a numeric constant. And we can create a character vector by setting its value to a string of characters surrounded by double quotes. There are a few other primitive data types, but these four are the ones that we're concerned with for the demos in our course. Once you have created variables you can display the value of these variables in the console by implicitly printing them. For example, our logical variable prints true or false as we would expect. Our integer variable prints as an integer, our numeric variable prints as a numeric value, and our character vector prints out a string of characters. Functions are first class citizens in the R programming language. We can create a function by using the function keyword in defining the arguments and the body of the function. For example, to create a function f with a single argument x and a body that returns x + 1, we type f set equal to a function with the argument x and a body that returns x plus the number 1. Now if we want to invoke that function, we simply type the name of the function, f, and pass in an argument for x. In this case we passed in the value 2, so 2 + 1 returns 3. In R we can assign functions to variables and pass them around like any other data type or we can use anonymous functions as well, that is, we can use the function without assigning it a named variable (Loading).

Data Structures Demo

In addition to the primitive data types and functions, we have more complex data structures in R as well. The first and most important of these data structures is a vector, which essentially a one-dimensional array containing elements of the same atomic data type. To create a vector, we use the concatenation operator, c, and pass in a comma separated list of homogenous data types. For example, to create a numeric vector containing the values 1, 2, and 3, we set v equal to a vector containing the values 1, 2, and 3. When we print the variable v we see a list containing the numeric values 1, 2, and 3. We can also create a vector of numerically ascending or descending values using the sequence operator, which is a colon. For example, to create a vector containing the values 1 through 5, we set the variable s equal to the values 1 through 5. When we print this variable s we see a list containing the numeric values 1 through 5 in ascending order. We can create a matrix, which is a special case of a two-dimensional array using the matrix keyword. For example, if we want to create a 2 x 3 matrix containing the values 1 through 6 populated in columnar fashion, we would set a variable m equal to a matrix containing the sequence of data 1 through 6 with 2 rows and 3 columns. When we print the variable m, we see a 2 x 3 matrix containing the values 1 through 6 populated in columnar fashion. We can create multidimensional arrays using the array keyword. Arrays can have one, two, or more dimensions. For example, if we want to create a 2 x 2 x 2 array containing the values 1 through 8, populated in columnar fashion, we would set a variable a equal to an array with the data 1 through 8 and set the dimensions arguments equal to a vector containing the values 2, 2, and 2. When we print the variable a we see a 2 x 2 x 2 array containing the values 1 through 8 populated in columnar fashion. In addition to creating data structures with homogenous data types, like vectors, matrices, and arrays, R also contains a special data structure called the list for storing lists of heterogeneous data types. That is, a list containing data of different data types. For example, to create a list of heterogeneous values in R, we could set a variable l equal to a list containing the logical value true, the integer 1, and numeric value, 2. 34, and the character string abc. When we print the variable l we see a list containing the values true, the integer 1, the numeric value 2. 34, and the character string abc. The formatting of the output of a list in the R console might look a bit messy and difficult to read. This formatting has to do with the way R is storing the values contained within the list. There is a special data type in R called a factor that stores an integer backed finite number of named values. This is similar to an enumeration in the C-like languages. For example, if we have a list containing the named values, male, female, male, male, female, we can store these values using a factor, which will store them as an array of integers mapped to each unique value. First we set a variable called factors equal to the factor keyword given a vector of strings. If we use the levels command, we will see the distinct list of name values in alphabetical order. The order in which these name values occur is the order that maps to the backing integers, for example, female maps to the integer 1 and male maps to the integer 2. If we use the unclass command, we can see the underlying integer array that is 2 1 2 2 1 and the two levels, that is female and male that compose the factor.

Data Frames Demo

A data frame is the most important data structure for exploratory data analysis. It essentially represents a table of data, that is it has a set of columns and a set of rows. Each column can contain a different data type, but all the data in the single column must be the same data type. We create a data frame using the data frame keyword. For example, we can set a variable df equal to a new data frame containing a column called name, populated with a vector of string values, cat, dog, cow, and pig. A second column called HowMany containing the vector of integers 5, 10, 15, and 20, and a third column called IsPet containing a vector of logical values true, true, false, and false. When we print our data frame we get pretty much what we would expect, a table containing three columns, that is Name, HowMany, and IsPet and four rows containing the respective values. Once we have our data stored in a data frame we can query and manipulate these data in extremely powerful ways. We can index our data frame by row and column using the index operator. For example, if we want to return the value contained in row 1 and column 2, we would type the following. As you can see, this returns the integer 5, which is the value contained in row 1 and column 2 of our data frame. In addition, we can omit the column argument and the indexing operation will return all values in the first row across all columns or we can omit the row argument and the indexing operation returns all values contained in the second column. We can also index on columns using the column name with the double square brackets notation. Or there's a syntactic shortcut that allows us to use the $ followed by the name of the column. All three of these examples return the same value, that is all rows in the second column. One of the most powerful language features for querying data in a data frame is called subsetting. Subsetting allows us to slice and dice data in very flexible ways. First we can pass a vector of integers indicating each row we want returned from the data frame into the rows argument of the index operator and it will return the specified rows. For example, passing a vector containing the integers 2 and 4 will return rows 2 and 4. In addition, we can pass a sequence of integers, that is 2 through 4, into the rows argument and it will return rows 2 through 4. We can also pass a vector of logical values, for example, true, false, true, and false, indicating which rows we want returned or not and it will return the corresponding rows, in this case row 1 and row 3. Although passing a vector of logical values by hand might seem like a trivial example, you can imagine how powerful this would be if the values in the vector were created programmatically using a predicate function. We can also subset using the equality operator. For example, we can return all rows where IsPet is true and we get the rows containing cat and dog. We can also subset using comparison operators. For example, we can query all rows where HowMany is greater than 10, which returns the rows containing cow and pig. We can also subset data tables with more advanced query expressions like finding all rows with values matching a list. For example, we can find all rows where the animal's name is contained in the list cat or cow. This will return just row 1 and row 3. And I should note that indexing and subsetting operations also apply to vectors, matrices, arrays, and lists in addition to data frames (Loading). A very important thing to note about R is that it's

Other Language Features Demo

A very important thing to note about R is that it's a vectorized language, this means that all atomic data types in R are a vector of size 1. In addition, most operations in R are vectorized, meaning that they take a vector as an input and return a vector as an output. There are, however, scalar, that is single valued equivalents for many of the vectorized operations in R. Now this might seem odd at first because it's very different from most languages you've probably worked with. However, it makes perfect sense in R because you're typically working with collections of data, for example, vectors, matrices, arrays, lists, and data frames. So working with them as collections is more practical and efficient. For example, if we want to add the values 1 and 2, we get the 3 like you would expect. However, this 3 is actually a vector containing a single element with the value of 3. In addition, if we were to add a vector containing the values 1, 2, and 3 to a vector containing the values 2, 4, and 6, we would get a resulting vector containing the values 3, 6, and 9. This is because the pairwise addition of 1 and 2 is 3, 2 and 4 is 6, and 3 and 6 is 9. Another important language feature to note is named versus ordered parameters. With R we can either pass arguments into functions using the name of the argument or we can pass arguments into functions by their default order. For example, we can create a matrix with the data 1 through 6 with the number of rows equal to 2 and the number of columns equal to 3, or we can create an identical matrix passing the same values by their default order. Now if we check to see if the elements of those two matrices are equal, we can see that we get a matrix back with the value true in all cells because the pairwise equality of each value in both of these matrices is true. Notice that the equality operator is vectorized as well. This is why the operation returns a matrix instead of a single scalar result. However, if you were interested in testing if these two matrices are identical, that is the scalar equivalent to the vectorized equality operation, we can see that the operator returns true as the two matrices are indeed identical. As I mentioned earlier in this module, R is very modular and extensible. It is very easy to download and install packages, which contain new functionality that extends the power of the language. For example, if you'd like to install a more powerful data visualization package called ggplot2, we could just type install. packages and specify the name ggplot2. When we press Enter, R will go online to the CRAN repository, download and install the ggplot2 package on our machine. Once the package has been installed, to use it in our code we simply type library and specify the package name we want to load. Now we have access to all of the commands and features found in the ggplot2 package. If you need help with any of the commands in R just type? and the name of the command you want help with. This will pop up the online documentation for that command in the help pane. From there, you can learn all about the commands, their arguments, return values, and more. Other than what we've seen in this demo so far, R contains most language features you'd expect from other C-like languages. Control structures like if, then, else statements, for loops, while loops, and switches, mathematical, relational, and logical operators, and string manipulation functions with regular expressions.

Summary

In this module we learned about the R programming language and why it has become so popular for data analysis. Next we learned about RStudio and that it is an integrated development environment or IDE for R. Then we learned about the basic language constructs, data structures, and ways to manipulate data in R. In the next module we'll learn how to transform and clean our data into a format suitable for exploratory data analysis.

Transforming and Cleaning Data

Overview

Welcome back to Exploratory Data Analysis with R. I'm Matthew Renze with Pluralsight and in this module we'll learn how to transform and clean our data to prepare it for exploratory data analysis. First we'll begin with an introduction to data munging or the process of transforming and cleaning our data. Next we'll learn how to load data into R. Then we'll learn how to clean and transform our data. Next we'll learn how to export data into various formats. Finally, we'll see a demo where we'll put all these steps together.

Introduction

Data munging is a popular term amongst data scientists for the process of transforming data from its raw form into a form suitable for data analysis. We do this because most data sets are not initially ready for analysis. Many data sets contain missing values, incorrect data types, incompatible units of measure, or incorrect variable encodings. So the data must be cleaned first to be in a format suitable for analysis. The term data munging is derived from the old computer jargon mung, which is an acronym for mash until no good, which is essentially the opposite of what we're doing when we're munging data. So while I'm personally not a huge fan of the term data munging, it appears to be the term most common in the industry for this process of transforming and cleaning data. Other terms that I've heard in the industry are data cleaning, data cleansing or data wrangling. When we're munging our data, we're typically doing several tasks including things like renaming variables, for example, correcting incorrect column names or removing invalid characters from column names. Data type conversion, for example, converting a character string to a numeric value or a numeric into an integer. Encoding, decoding or recoding values, for example, converting male and female to the letters m and f or vice versa. Merging data sets, for example, joining two tables of data based on a key that is shared between the two tables. Converting units, for example, converting mils to kilometers or scaling grams to kilograms. Handling missing data, for example imputing data, that is replacing missing values with substitute values. Handling anomalous data, for example, identifying and correcting statistical outliers created by data entry mistakes, and many more tasks. The first step in data munging is loading the data into R. Luckily for us, R supports a wide variety of data sources including file-based data, for example, comma separated values; tab delimited values, and Excel files. Web-based data, for example, XML, HTML, and JSON data. Databases, for example, SQL Server, Oracle, and MySQL. Statistical data files, for example, SASS, SPSS, and Stata files, and many more. In fact, there are literally hundreds of data sources that R supports using downloadable extension packages. The second in data munging is transforming and cleaning our data. Unfortunately this step is often the most difficult and the most time consuming. So my recommendation is to record all steps using a script so that you can reapply those steps whenever they are needed, because it's inevitable that you'll be on step 50 of a data analysis and realize that you've made a mistake back in step 5. You don't want to have to go back and reapply every step by hand. In addition, once you've performed a data analysis, it's likely that you'll be asked to perform it again sometime later with the latest data. If you have all the data transformation steps recorded as an executable script, you just point the script at the new data and run the analysis with the new data. So if our goal in data munging is to end up with clean data, it's probably a good idea to know what clean data look like. There are two main aspects of clean data. The first is the structure of the data. By the structure of the data being clean we mean that each table contains one and only one type of observation, transaction, or entity, for example, temperature readings from a sensor, sales at a pizza shop, or movies released in theaters. There should be one column for each variable, that is values that vary across each observation. For example, the data column should contain the data of the observation and the quantity sold column should contain the number of items of a specific product sold to a specific customer on that specific date. Each of these variables or columns should have a human readable name, like Date, Customer, Product, or Quantity sold. The observations should be stored in rows with one row for each observation, transaction, or entity. Finally, each row should be uniquely identified using a natural key, a surrogate key, or a combination of columns. The second aspect of clean data are the data themselves. Ideally we want our data to contain no errors, that is all values should be correctly recorded. There should be no missing values, that is, there should be a value for each variable in each observation. They should be properly encoded, that is the encoding scheme for each variable should be consistent across all observations, and all variables should be internally consistent across each observation, that is each value should have the same data type, unit of measure, and scale. When we have clean data our job is much easier and the results of our analysis are much more likely to represent reality. While it may not be necessary for exploratory data analysis, it is oftentimes necessary to export data from R into a target data format of some kind. Once again, R makes this very easy by supporting a wide variety of export formats, including file-based formats, web-based formats, databases, and statistical data files.

Demo Setup

For our data munging code demo, we're going to be working with a data set from the open movies database containing a selection of movies released in the US between the year 2000 and the summer of 2015. We will be only including movies that have a rating of G, PG, PG-13, or R that have a reported box office revenue. In order to keep things interesting though, we're going to create a fictional narrative to help motivate the remainder of our demos. Imagine for me if you will that we work at a Hollywood movie studio. However, this studio doesn't produce blockbuster movies; they produce those really bad B movie versions of major blockbuster movies. In addition, we have a coworker named Joe who works in the finance department of this movie studio. One day Joe comes to us in a panic. He says to us, I have this report that the CEO needs finished by the end of the day, but I can't load the data file into my analysis software. I'm trying to load a data file containing data about thousands of movies, but there are a bunch of problems with the file that are causing it to crash my analysis program. There's a column with the wrong name, something about a rotten tomato, there are rows with missing values, the runtime column contains the abbreviation for minutes in each row next to the actual runtime, the box office revenue column contains revenue in whole dollars, thousands of dollars, and millions of dollars, and the file uses tabs instead of commas to separate the values in the file. Joe is convinced that he's going to get fired if he can't finish this report by the end of the day, so he doesn't know what else to do. Joe knows you're a programmer and he thinks you're a pretty smart guy or gal so he comes to you asking for help. So you say to Joe, you know Joe, I watched this Pluralsight video last week and I learned about a programming language called R. I think I might be able to help you out. So we're going to help Joe clean up his data file so he can finish his report for the CEO by the end of the day.

Loading Data

First, let's take a look at Joe's source file with a text editor like Notepad++. We can see that it is a tab delimited file format because it uses a tab character represented by the red arrows to separate each field within the row. In addition, we can see that it uses a carriage return lined feed character sequence represented by the CR and LF blocks to specify the end of each row. The first row in the file contains headers, which are our column names. We have six columns, Title, Year, Rating, Runtime, Tomato Meter, and Box Office. In addition, some but not all the movie titles are surrounded in double quotes. There appears to be double quotes when the movie contains a special character that might interfere with the parsing of the data file. Once we're in RStudio, the first thing we need to do is to set the working directory. This is the folder that contains the data file that we want to load. We do this using the setwd command and specifying the path of the folder we want to set as our working directory. Notice that this path is a Unix style folder path, that is, it contains forward slashes rather than the Windows style folder path, which would contain backslashes. Joe's analysis software was expecting a CSV, that is comma separated values file, but instead someone gave him a tab delimited file. R allows us to load data sources in very flexible ways. For example, to load a tab delimited data file, we use the read table command, set the file to movies. text, set the character separator to the tab character using the character escape sequence\t, indicate that the file contains headers, and set the quote character to a double quote, again, using a character escape sequence because the file wrapped some of the character strings in double quotes. When we press Enter, this will load the data into a data frame assigned to a variable named movies. The first thing I like to do after loading a data source is to take a quick peek at the data using the head command and specifying the name of the data frame, in our case, movies. This will show us just the first six rows of data in the data frame. As we can see in the console, we have data corresponding to movies released in the US after the year

2000. We have the movie title, the year it was released, ratings, that is G, PG, PG-13, or R, movie runtime in minutes, something called a tomato meter, which Joe tells us is the movie critic score derived from the rotten tomatoes website and box office revenue with dollar sign prefixes and an uppercase m suffix for millions and lowercase k suffix for thousands. The next thing I like to do is inspect the column names. I like to do this early in the data munging process because I frequently need to know the names of the columns and it's nice if I have them on hand somewhere. As you can see, the names command displays the names of the six columns in our data frame. The first problem that Joe described is that there's a column with an

Cleaning Data

The first problem that Joe described is that there's a column with an incorrect name. We can see here that we have a column named Tomato Meter and Joe's analysis software is expecting it to be titled Critic Score. We can easily rename this column by setting the name of the column at index 5 to Critic Score, and problem 1 is solved. R makes it very easy to make structural changes to the data, like renaming a column. The second problem that Joe describes is that his data file has a few rows containing missing values. R represents missing values as na values, which are equivalent to null values in other languages. First, we can determine how many missing values are contained in a data frame by getting the sum of the na values using the is. na command. As we can see, we have four missing values in the rows of our data frame. I've intentionally placed these rows with missing values at the end of the file so we can quickly take a look at them. We'll do this with the tail command, which does the opposite of the head command, that is, it returns the last six rows of the data frame. Here we can see our four rows containing NA values. Typically we would want to handle missing values in a way that maintains the statistical validity of the data, like imputing the missing values with substitute replacement values. However, for Joe's specific case we're going to just eliminate the rows containing missing values. We can do this with the na. omit command. And we can see, this has eliminated the rows with the missing values because the sum of the is. na command now returns 0. The third problem that Joe described is that there's an abbreviation for minutes in the movie runtime column. Using the head command on the runtime column we can see what Joe is describing. If we try to run a mathematical operation on these data, we can see that the operation fails because the value is not integer since it contains text. We're going to remove the unit of measure for each row and convert this column into an integer data type. First, let's run the class command on the column to inspect its data type. We can see here that the data type is a factor. This is because R loaded this value as a factor because it contained text. Next, we'll cast these factors into character strings by using the as. character function. We can see that these values have been converted to character strings because they are now wrapped in double quotes. We can verify the data type by using the class command and we see that this column is in fact a vector of type character. Now that we have these data represented as character strings, we need to remove the abbreviation for minutes and the extra white space. We can do this with a substitute command. We pass in a regular expression indicating the pattern of text that we want to replace and the replacement value, which in this case is an empty string. Regular expressions are a very powerful language for textual pattern matching that is built into R and many other programming languages. We can see now that the minute's abbreviation and the extra white space has been removed from all values in this column. Finally, we want to convert the character string into integer values using the as. integer command. As we can see, the values are now numeric since the double quotes have been removed. We can verify this by executing the class command and we see that the column is in fact a vector of integers. Now we can perform mathematical operations on the values in this column. The fourth problem that Joe described was that the Box Office column contains revenue in whole dollars, thousands of dollars, and millions of dollars. If we run the head command on the column, we can see that all of the values contain a $ prefix followed by a numeric value and then either an uppercase M suffix for millions, lowercase k suffix for thousands, or no suffix for whole dollar units. In order to fix this problem we're going to create a function that converts these factors into character strings, removes the $ prefix and the uppercase M or lowercase k suffixes, converts the value into a numeric, and finally scales that numeric value from either dollars or thousands of dollars into millions of dollars. So, we'll end up with a numeric column containing all values in millions of dollars. First we'll start by declaring a function called convert box office that takes the value called boxOffice as an argument. Next, the function will convert the box office factor into a character string. Then the function will replace the $, uppercase M and lowercase k with an empty string. Next, the function will convert the replaced string to a numeric value. Finally, the function will scale the value to millions of dollars based on the suffix that was contained in the character string. We do this by using the grepl function with a regular expression to determine if a character string contains an uppercase M or a lowercase k. Grepl gets its name from the Unix command grep whose name is an acronym for Globally Search a Regular Expression and Print. The L at the end of the name stands for Logical, indicating that this is a logical or Boolean equivalent to the regular grep command, that is, it returns true or false based on whether a match is found or not. So if the character string contains an uppercase letter M, then we just return the numeric value since the value is already in millions of dollars, else if the character string contains a lowercase k, then we return the numeric value multiplied by 0. 001 to scale the value to millions of dollars. Otherwise, if there is no suffix we return the numeric value multiplied by 0. 000001 to scale the whole dollar values to millions of dollars. Now that we have our function we can apply it to all values in the Box Office column using the sapply function. Sapply is one of the many apply functions in R that allows us to apply a function to each element in a collection of values. We type sapply, then specify the collection we want to apply a function to followed by the function we want applied and sapply will execute that function for each item in the collection, passing the item in as an argument into the function. So in this case, we are applying the convertBoxOffice function to each box office revenue in the Box Office column, passing each box office revenue into the function one at a time, returning a vector containing the results. Then we are assigning the resulting vector of numeric values in millions of dollars back to the Box Office column. We've now solved problem four. We can use the head command to see that the values are all numeric and scaled to millions of dollars. Next, we can verify that the column is in fact a vector of numerics using the class command. Finally, we can now execute mathematical operations on the values in this column.

Exporting Data

The last step in our demo is to export our data frame into a comma separated values file so that Joe can load it into his analysis program. We do this using the right CSV command specifying our data frame as a source and the name of the new CSV file that we want to create. Let's open our output file in a text editor like Notepad++. We can see that the file is a comma separated values file because the fields are now delimited using a comma character. In addition, the end of each row is still specified with a carriage return line feed character sequence. Our first row contains headers, that is our column names, surrounded in double quotes. We have our six columns, Title, Year, Rating, Runtime, Critic Score, and Box Office, plus an additional column on the left, which contains the index of each row in the data frame. R included this extra ID column by default. In addition, R replaced empty spaces in our column names with a period character by default. All of our character strings, that is titles and ratings, are surrounded by double quotes while our integer and numeric values have no double quotes. All in all this is a pretty good looking CSV file.

Demo Conclusion

So we hand the CSV file to Joe, he runs his analysis, completes his report, hands it to the CEO, and we've saved the day. Joe doesn't get fired. In fact, the CEO is so impressed that they promote him to CFO. So Joe takes us out to the bar to celebrate our success and his new promotion.

Summary

In this module we learned about data munging or the process of transforming and cleaning our data into a format suitable for analysis. Next, we learned how to load data into R. Then we learned about clean data and how to transform and clean our data. Next, we learned how to export our data from R into other formats. Finally, we saw a demo that tied all these steps together. In the next module we'll learn how to calculate descriptive statistics for our data.

Calculating Descriptive Statistics

Overview

Hello again and welcome back to Exploratory Data Analysis with R. I'm Matthew Renze with Pluralsight and in this module we'll learn how to calculate descriptive statistics for our data. First we'll begin with an introduction to descriptive statistics, that is describing the characteristics of our data in meaningful ways. Next we'll learn about the types of analysis that we can perform on our data. Then we'll learn how to avoid making errors and invalid claims with our descriptive statistics. Finally, we'll see a demo where we'll put all these concepts together.

Introduction

Descriptive statistics describe data in meaningful ways. In exploratory data analysis, we're typically trying to quickly assess the location, spread, shape, and interdependence of our data. These types of descriptive statistics are often referred to as summary statistics because they summarize the shape and feel of the data. For example, in this table, we have the summary statistics for a single variable, that is the movie runtime variable in our movies data set. We have the Minimum, which is the lowest value in the column when the values are sorted in ascending order. First Quartile, which is the value that cuts off the first 25% of the values, the Median which is the value that separates the lower half from the upper half of the values, the Mean which is the arithmetic average of all of the values in the column, 3rd Quartile which is the cutoff value for the 75th percentile of the values, and maximum which is the highest value in the column. By looking at these summary statistics we can quickly learn about the location and spread of our data. We don't want to go too deep into statistics for this course, but in order to discuss descriptive statistics we need to understand a few basic terms from statistics. First we have observations, which are essentially the rows in the table. They are referred to as observations because in statistics we're typically concerned with observations of some kind of physical phenomenon. For example, if we have a temperature sensor, each recorded temperature over time would correspond to an observation of the temperature. Equivalently, we could also have transactions, like pizza sales transactions, or entities, like feature length films as the phenomena we are observing. However, for our purposes, we'll refer to them all generically as observations. Next, we have variables, which are the columns in the table. They are called variables because their values vary across each observation. For example, in the table to the right, Date, Customer, Product, and Quantity are all variables that can change value across each row in the table. There are two types of variables, first we have qualitative variables. Qualitative variables contain categorical values, for example, customers and products. In addition, they have no nature sense of order. We can, however, impose an arbitrary order upon them, like using the alphabetical sort order of their names. However, this is just an artificial, not a natural means of sorting them. Qualitative variables are often referred to as nominal variables because they are named values. Finally, we have quantitative variables. Quantitative variables contain numeric values, for example, the quantity of products sold. In addition they do posses a natural sense of order, for example, 2 pizzas sold is more than 1 pizza sold. Quantitative variables can also be subdivided into either discrete values, that is whole numbers represented as integers, or continuous values, that is all possible points on the real number line, typically represented as decimal precision numeric values. In addition, quantitative variables can also be subdivided into ordinal, interval, and ratio subtypes. However, these subdivisions, their differences, and statistical limitations are outside of the scope of this course. There are several types of statistical analysis we can do, which are dependent upon the number of variables. We can perform univariate, that is single variable analysis, or bivariate, that is two variable analysis, and the type of variables, whether we have qualitative, that is categorical values, or quantitative, that is numerical values. We'll dig deeper into each of these types of analysis next (Loading).

Univariate Analysis

The first type of analysis that we can perform is univariate analysis of a qualitative variable. This is just a fancy term for the analysis of a single categorical variable. When we're performing qualitative univariate analysis, we're typically interested in determining the frequency of observations that occurred in a specific category. For example, if we were interested in the number of movies that are comedies in our data set, we would find this value by scanning down the table to the Comedy row and inspecting the frequency. In addition to frequency, we might also want to know how the number of observations relates to all observations, that is the part of the whole. This value is typically expressed as a percentage. The third type of measure we might be interested in is the mode, that is category containing the most observations of the variable. In the case of our table of movie genres, we can see that drama is the category most frequently observed in the data set. There are other measures that we might be interested in while performing qualitative univariate analysis, but these should cover the basics. The second type of analysis we can perform is the quantitative univariate analysis, which is just a fancy term for the analysis of a single numeric variable. When we're performing univariate analysis, we're typically concerned with the location, that is measures of central tendency; spread, that is measures of dispersion; and the shape of the data for a single numeric variable. We'll take a look at each of these types of measures next. When we're interested in knowing the location of the data we're typically looking at measures of central tendency. For example, the mean, which is the arithmetic average of all of the values observed for the variable; the median, which is the value that divides the upper and lower half of the data or the average of the two middle values if there's an even number of values; and the mode, which is the most frequently occurring value observed for the variable. When we're interested in the spread of the data, we're typically looking at measures of dispersion. For example, the minimum value observed for the variable; the maximum value observed for the variable; the range, which is the difference between the minimum and the maximum values; the quartiles, which are the points where the data are divided into four equal partitions; the variance which is a measure of how far the values are spread out; and the standard deviation, which is the square root of the variance so that the values are all scaled to the same dimension as the observations. When we're interested in the shape of the data, we're typically looking at the skewness, which is a measure of the asymmetry of the distribution of values and the kurtosis, which is a measure of how sharply peaked or relatively flat the distribution of values are. Both of these values are measured relative to the normal distribution, which is a distribution with a bell shaped curve, which models the distribution of probabilistic outcomes of many physical processes in our world. There are many other measures that we can obtain for univariate analysis, but these should cover the basics.

Bivariate Analysis

The third type of analysis we can perform is qualitative bivariate analysis, which means the analysis of two categorical variables. When we're performing qualitative bivariate analysis, we're typically interested in the joint frequency of the observations, that is the frequency of occurrences of the observations at the intersection of two categories, for example, movie genre and movie ratings. We can display bivariate frequency distributions using a contingency table, which is a table in matrix format containing the frequency of observations at the intersection of the rows and columns. For example, in this table, if we're interested to see how many PG rated comedies there are in our data set we would find the value contained at the intersection of the comedy row and the PG column. Our contingency tables can also contain percentages instead of frequency counts. The percentages in each cell of the table would sum to 100%. In addition, our contingency table can contain row totals on the right hand margin of each row showing the sum of all observations for the rows and column totals on the bottom margin showing the sum of all observations for each column. A contingency table is often referred to as a frequency table, a two way table, or a cross tabulation matrix. The fourth type of analysis we can perform is quantitative bivariate analysis, which means the analysis of two numeric variables. In quantitative bivariate analysis, we're looking at the relationship between two numeric variables. For example, how does one variable influence, predict, or correlate with a second variable? In this course we will only be looking at correlation, since we'll be focusing on descriptive statistics rather than predictive statistics. Predictive statistics, which is outside of the scope of this course, looks at how one variable influences another variable or how strongly one variable predicts another variable. In the chart on the right we have two variables pertaining to Old Faithful, one of the most predictable geysers on earth located in Yellowstone National Park. On the X axis we have the duration of the eruption in minutes, this is the independent or predictor variable. On the Y axis we have the duration of time before the next eruption in minutes, this is the dependent or outcome variable. There is a relationship of interdependence between the eruption duration and the waiting duration that we would describe as a strong positive correlation between these two variables. When we're performing bivariate analysis we're typically looking at measures of the interdependence of the two variables. For example, the covariance, which is the degree to which the two variables vary with one another, and the correlation coefficient, which is covariance on a standardized scale so that they can be compared across variables in an apples to apples way. There are also more measures that can be used for bivariate analysis, but once again these two should cover the basics. The fifth and last type of analysis we will cover is bivariate analysis for both a qualitative and a quantitative variable. This means that we're looking at the relationship between a numeric variables values across multiple categories. Essentially we're using the quantitative measures from our quantitative univariate analysis, partitioned by categories so that we can compare them from one category to the next. For example, if we're interested in comparing the average critic score across all movie genres, we could create a table of each genre and its respective average critic score. In our table we can see that the critics rated animated movies higher than horror films in general. Once again, there are many more measures we could look at, but this should get us started.

Guidance

Before we see how to perform these various types of analysis in R, I think it's very important that we spend a bit of time discussing how to avoid making errors and invalid claims with our analysis. This is a critical point of discussion because mistakes in the source data, in analysis or a claim can have serious consequences for you, your company, and your reputation. First, we want to make sure our data are as clean as reasonably possible before performing our analysis. There's an old saying in the computer programming world, garbage in, garbage out. If we're starting with unreliable data we can likely expect our results to be unreliable as well. Second, it is important that we understand the domain of the data we are analyzing. For example, if we're working with data for insurance, it's important to understand what a policy is, what deductibles, premiums, and co-insurance are, what units they are expressed in, and whether they can be averaged, summed, divided, et cetera. Coincidentally this is why all of the examples and demos in our course have involved very simple domains like pizza sales transactions and feature length movies. These domains are both simple and most people understand them relatively well. Essentially without understanding the context around the data, it's very easy to make errors based on that lack of understanding. Third, it's important to understand biases and how they influence the results of our analysis. There are many types of biases ranging from cognitive biases, statistical biases, and contextual biases. I encourage you to learn about them and countermeasures you can take to protect your analysis as best as reasonably possible from their unintended consequences. Fourth, we should strive to make our data analysis reproducible. In the event someone questions our results, we should be able to provide them with all of the data and all of the steps involved in the analysis. This way they can inspect the data, inspect the analysis steps, and rerun the analysis to determine if there were any mistakes in our data or our analysis that could have produced incorrect results. Fifth, it's important to understand the implications of the results of our analysis. For example, if we show these results to our boss, will they be making critical business decisions based upon them? We should always strive to understand the impact of our results and invest effort in our analysis in proportion of the magnitude of the possible impact of these results. Finally, and this is most important of all, you must recognize your limitations. Please do not attempt to perform analysis beyond your competency without the assistance of someone who has the requisite, knowledge, training, and experience. Statisticians and scientists have many years of education and hands on training and experience regarding the collection, analysis, and interpretation of data in very rigorous ways to minimize the likelihood of errors. And even with all this extensive education and training they still make errors from time to time. Ultimately, please do not practice advanced statistical data analysis without the proper training. The necessary training is a graduate degree in statistics, data science or related field with a research or quantitative analysis component. Performing day to day analysis of data for software developer tasks, like analyzing log files, performance metrics, and code quality metrics, is well within the competencies of most software developers. However, advanced statistical analysis that may have significant economic, legal, or health implications should only be performed by properly trained individuals. Think of it like medical training, the average person is more than qualified to apply antiseptic and bandages to a wound, but only a trained heart surgeon should ever attempt to perform open heart surgery (Loading).

Demo Setup

So our coworker Joe is now chief financial officer for our low budget Hollywood B movie studio. He really wants to impress the board of directors and show them that he's got what it takes to make good financial decisions, so he comes up with an idea for a new movie. Keep in mind our movie studio's only concern is making a profit, they could care less whether the movie provides any value to society or not. They only care that it generates ticket sales. While we're having drinks together at the bar, Joe tells us that he's come up with a great idea to make a new, highly profitable, low budget B movie for the studio. He says he's been watching a lot of movies lately and he thinks that the movie industry is just dying for an Avant-Garde sci-fi western musical that's an R rated, 3 hour epic. He says the critics will hate it, but the audiences will love it, and he predicts that the box office revenue will be off the charts. This thing is going to be bigger than gigantic, the low budget rip off we created of the Hollywood Blockbuster movie, Titanic. Well we're a sciencey kind of person and so we're a bit skeptical of Joe's claim. In addition, we just so happen to have our laptop and the movies data set at the bar with us, like any good programmer would. So let's try and see if Joe's claim holds any water by determining which types of movies brought in the highest box office revenue from 2000 to 2015, and if Joe's Avant-Garde, 3 hour, R rated, sci-fi, western musical might be a box office hit or sink faster than the RMS gigantic. First, let's start again by setting our working directory to the folder containing our source data file. Next, we'll load our CSV file, movies. csv, using the read. csv command. We'll also specify the double quote character as our quote character since the character strings are wrapped in double quotes. In addition to the movies data set we're also going to be loading a second data set containing move genres. For example, genres like action, adventure, comedy, and drama. Since most movies fall into more than one genre, this file contains one row for each movie for each genre tag. For example, the 2000 movie, Gladiator, is listed as both an action movie and a drama, so there will be two records for this movie, one for each genre it belongs to. We will mainly be using the movies data set, but there are few tasks that we'll perform with the genre data set. Now let's take a peek at the movies data set using the head command. We can see that we have all the columns and data that we would expect. Let's also take a look at the genre data set using the head command as well. As you can see, we have one row for each movie and for each genre tag. Notice that the last two rows contain both the action and drama records for the movie Gladiator, just like we described earlier.

Univariate Analysis Demo

The first type of analysis we can perform is univariate analysis of a qualitative variable, that is, analysis of a single categorical variable. We'll do this using the table command on a categorical variable like the movie rating. As we can see, this returns a table of the frequency of occurrences of movies in each rating category. For example, we have 93 G rated movies, 497 PG movies, 1235 PG-13 movies, and 1423 R rated movies. Let's do the same analysis for movie genres as well. As we can see, we have a large number of action movies and dramas, however, we have a relatively small number of musicals and westerns. Next we'll perform univariate analysis of a quantitative variable, that is, analysis of a single numeric variable. First we'll look at ways to measure the location of the data, that is, measures of central tendency. We'll use the movie runtime in minutes for our quantitative variable. Let's start with the mean, which is the arithmetic average of all movie runtimes. As we can see, the average movie runtime is app 104 minutes. Next we'll look at the medium runtime, which is the value that divides the upper and lower half of the data or the average of the two middle values if there's an even number of values. As we can see, the medium runtime is 101 minutes, which is slightly less than the mean, so the shape of this distribution of values will likely be positively skewed, that is right skewed. However, this is not always the case since there are exceptions to this guideline, so it's not an absolute rule. Finally, we'll look at the mode of the runtimes, which is the number of minutes of runtime most frequently occurring in the data set. Unfortunately there is no built in command in R to inspect the mode of a quantitative variable, but we can quickly determine it by creating a frequency table of the values and using the which. max command to return the runtime with the highest frequency. As we can see from the results, 90 minutes is the most frequently occurring runtime in the data set. The 27, which is also returned by the which. max function, is the index in the frequency table where the entry for the movies with 90 minute runtimes occurred. Second, we'll look at ways to measure the spread of the data, that is, measures of dispersion. First, we'll inspect the minimum, which is the shortest movie runtime in our data set. As we can see, the shortest movie runtime is 38 minutes. Next, we'll inspect the maximum value, which is the longest movie runtime in our data set. As we can see, the longest movie runtime is 219 minutes, which is just over 3 1/2 hours long. Then, we can inspect the total spread of the data by looking at the difference between the minimum and the maximum values. We'll do this by getting both values as a vector using the range command, then we'll apply the diff command to the resulting vector to get the difference from the minimum and maximum values, which as we can see, the difference is 219 minutes minus 38 minutes, which is 181 minutes. Next, we'll take a look at the quartiles of the movie runtime values, which are the cutoff points where the data are divided into four equal partitions. We'll do this using the quantile command, which returns a table containing the five cutoff points for the four partitions. If we want to grab an individual cutoff point, we can pass the cutoff point into the command and return just that value. For example, we can inspect the 25th percentile, that is the cutoff point at the end of the first quartile, which is 93 minutes. We can also get more fine grained than quartiles by using percentiles, that is the cutoff points where the data are divided into 100 equal partitions. For example, we can inspect the 90th percentiles cutoff value, which is 126 minutes. In addition, if we're interested in inspecting the spread of the data using the quartiles, we can calculate the inter-quartile range using the IQR command. This returns the third quartile, which is 113 minutes minus the first quartile, which is 93 minutes. It's essentially the difference between the middle 50% of the values. As we can see, there's a difference of 20 minutes between the first quartiles cutoff point and the third quartiles cutoff point. Next, we can inspect the variance of the data, which is a numerical measure of how far the data are spread out. We do this using the variance command. As we can see, we have a variance of app 284 minutes squared. The smaller the number, the smaller the spread. The larger the number, the larger the spread. However, minutes squared is essentially a meaningless unit of measure for comparison purposes. But we can take the square root of the variance to square the values back down to minutes. We do this using the standard deviation command. Here we can see that one standard deviation is approximately equal to 17 minutes. Once again, the smaller the number the smaller the spread, the larger the number the larger the spread. Third, we'll look at ways to measure the shape of the data, that is measures of skewness and kurtosis. Unfortunately there's no built in commands in R to measure skewness and kertosis, but luckily there's an extension package called moments that contains these features. We'll download this library using install. packages and pass moments in as an argument, and then we'll load the moments package using the library command. First we'll look at the skewnes, we'll do this using the skewness command. It returns either a negative or positive value indicating the direction of the skew. If the skewness is 0, then the distribution of data is perfectly symmetric. If the skewness is negative, then the distribution of data is negatively skewed, that is the tail is skewed to the left. Otherwise, if the skewness is positive then the distribution of data is positive skewed, that is the tail is skewed to the right. In addition, the larger the value the greater the skew is in either the negative or positive direction respectively. Our movie runtime has a skewness of approximately 1, so the data are positively skewed, that is a right skewed distribution. Next we'll look at the kurtosis, that is peakedness, to determine how sharp or how flat is the peak of the distribution of data. We'll do this using the kurtosis command. It returns a value that is either greater than, less than, or equal to 3. A value of 3 means the shape of the peak of the distribution of data matches the normal distribution. A value of less than 3 means the peak is flatter than the normal distribution. A value greater than 3 means the peak is steeper than the normal distribution. As we can see in our movie runtime data, the kurtosis is greater than 3, so the peak is steeper than the normal distribution. To confirm both our skewness and our kurtosis values, let's create a quick plot of the distribution of movie runtime values. As we can see, in the graph in the lower right hand corner, the data are in fact positively skewed, that is the tail is skewed to the right, and the peak appears to be steeper than that of the normal distribution. Don't worry about the commands I used to plot this graph right now, we'll do plenty of data visualization in the next module. If you'd like to quickly summarize a single quantitative variable, you can use the summary command. It returns the 5 number summary, that is minimum, lower quartile, median, upper quartile, and maximum in addition to the arithmetic mean. This is a great way to quickly get a feel for the location and the spread of the data.

Bivariate Analysis Demo

The next type of analysis we will perform is bivariate analysis of qualitative variables, that is analysis of two categorical variables. To perform this type of analysis we'll build a contingency table to display the frequency of observations at the intersection of each category of both variables. For example, if we want to see the frequency of occurrences for each movie genre across each movie rating category, we can use the table command and specify the two qualitative variables we want to analyze. As we can see, our contingency table is a two-dimensional matrix containing genres on the rows and ratings on the columns. Each cell in the matrix is populated with the number of movies tagged with each genre and possessing the corresponding rating. For example, we can see that there are 43 G rated movies tagged as animated films and there are 836 R rated movies tagged as dramas. The next type of analysis we will perform is bivariate analysis of quantitative variables, that is analysis of two numeric variables. To perform this type of analysis, we'll first look at the covariance of the variables, that is the degree to which the two variables vary with one another. We do this with the covariance command. As we can see, the two variables vary with one another in a positive way. That is, larger values of the first variable correspond to larger values of the second variable in general. If this value were negative, then the two variables would inversely vary with one another. That is, large values of the first variable would correspond to smaller values of the second variable in general. In this example we can see that the movie runtime and box office revenue have a positive covariance, so in general as the movie runtime increases, the box office score also increases. We'll also take a look at the covariance of critic score and box office revenue. As we can see, the critic score and box office revenue also have a positive covariance, so in general as the critic score increases the box office revenue also increases. However, we cannot compare these two covariance values to determine whether the runtime or critic score correlate more strongly with the box office revenue. In order to compare the correlation in an apples to apples way, we need to use the correlation coefficient. Correlation coefficient is covariance on a standardized scale. For example, -1 is a total negative correlation, 0 is no correlation, and +1 is a total positive correlation. We can inspect the correlation coefficient of two quantitative variables by using the correlation coefficient command. As we can see, the correlation coefficient of runtime and box office revenue is approximately 0. 34 and the correlation coefficient of critic score and box office revenue is approximately 0. 16. This means that movie runtime more strongly correlates to box office revenue than critic score because the first value is larger than the second value. However, it is important to note that both of these correlation coefficients are relatively small so neither would be suitable as a predictor for box office revenue. In addition, this value only tells us that the two variables correlate with one another, it says nothing about whether one variable influences the outcome of the other variable. Correlation does not imply causation. So it looks neither movie runtime nor critic score will be useful in determining whether Joe's 3 hour epic that critics will hate is likely to be a box office hit or not. The last type of analysis that we will perform is bivariate analysis of both qualitative and quantitative variables, that is the analysis of one categorical and one numeric variable. When we're performing this type of analysis we're typically looking at a quantitative measure across a set of categories. For example, we might want to know what the average box office revenue is across each movie rating category. To do this, we'll use the tapply command, which is another one of the many apply commands in R. Much like the sapply command, the tapply command allows us to execute a function over a collection of values. However, unless the sapply command, tapply also allows us to group those values by a factor, for example, movie ratings or movie genre, before applying the function. So if we want to see the average box office revenue across each movie rating category, we type tapply, pass in the value that we want to apply a function to, and the factor we want to use to group the values, and finally the function we want applied. In this case, we're applying the mean function to the box office revenue grouped by each rating category. As we can see, PG movies in general had the highest box office revenue followed by G movies, then PG-13 and finally R rated movies. So Joe's decision to make his movie R rated might not be the best idea if his only goal is to maximize box office revenue. In addition, we'll do the same analysis to determine average box office revenue for each movie genre. As we can see, Action, Adventure, and Fantasy movies all did very well at the box office. Sci-Fi does relatively well too, however, Westerns and Musicals did not do very well at all. So once again, Joe's decision to make a sci-fi western musical may not be the best idea if his sole goal is to maximize box office revenue.

Demo Conclusion

Finally, if we'd like to summarize an entire table of data all at once, we can use the summary command and pass in the data frame as an argument. As we can see, this gives us summary statistics for each of the variables in our data frame. In addition, we are given a set of summary statistics most appropriate for the type of variable. Qualitative variables, that is categorical variables, are summarized as a frequency table and quantitative variables, that is numeric values, are summarize as 5 number summaries and an arithmetic mean. For example, movie titles and movie ratings are displayed as frequency tables, however, Year, Runtime, Critic Score, and Box Office revenue are all displayed as 5 number summaries and the arithmetic mean. Summarizing an entire table can be a quick and easy way to learn a great deal about a data set with minimal effort. So with our back of the napkin guestimate, we conclude that a PG rated, fantasy, action, adventure movie might be a safer bet than Joe's 3 hour, R rated, sci-fi, western musical, provided that Joe's only goal is to maximize box office revenue. So we walk Joe through our analysis, he realizes that we may have just saved him from another big mistake, and he buys us another round of beer to celebrate. In addition, it's important to note that we're having a bit of fun with this analysis. While we can use our movies data set to say what happened in the past, attempting to predict whether a movie will perform well at the box office is a more complex task and well beyond the scope of the analysis we'll be performing in this course.

Summary

In this module we learned about descriptive statistics and how they can be used to describe our data in meaningful ways. Next, we learned about the various types of analysis that we can use to describe our data. Then we learned how to avoid making errors and invalid claims with our descriptive statistics. Finally, we saw a demo that tied all of these concepts together. In the next module we'll learn how to visualize our data using the basic plotting system in R.

Visualizing Data

Overview

Welcome back to Exploratory Data Analysis with R. I'm Matthew Renze with Pluralsight and in this module we'll learn how to visualize our data with the basic plotting system in R. First we'll begin with an introduction to data visualization, that is representing the characteristics of our data in visual ways. Next, we'll learn about the types of data visualizations that we can create for our data. Then we'll learn how to create clean data visualizations and avoid making mistakes with our visualizations. Finally, we'll see a demo where we'll put all these concepts together.

Introduction

Data visualization is the representation of data via visual means. We do this because the human brain is exceptionally good at visual pattern recognition, so we are able to quickly learn about a set of data by transforming it from its raw textural numeric form into a form that can be inspected visually. The essential idea is to map the dimensions of our data, that is the variables, to visual characteristics. For example, if we take a look at this table of data it would takes us a bit of time to determine whether we sold more pizzas or more salads during the dates listed in the table. However, if we represent these data as a bar chart, we can quickly see that we sold quite a bit more pizzas than salads. Just by representing our data visually, we can recognize patterns in our data more quickly than if we were to perform a numeric analysis. Once again, there are several types of visual analysis we can perform, which are dependent upon the number of variables. We can perform univariate, that is single variable visual analysis, or bivariate, that is two variable visual analysis, and the type of variables, whether we have qualitative, that is categorical variables, or quantitative, that is numeric variables. We'll dig deeper into each of these types of visual analysis next.

Univariate Analysis

The first type of visual analysis we can perform is univariate analysis of a qualitative variable, that is the visual analysis of a single categorical variable. When we're performing a visual analysis of a categorical variable, we're typically interested in comparing the frequency of observations that occurred in a specific category. We can quickly compare the frequency of observations for sets of categories by using a vertical bar chart, also known as a bar graph or a column chart. In this chart, we have the frequency of observations of movies by their movie rating category. On the X axis we have movie rating categories, on the Y axis we have a count of the number of movies released. The height of each bar represents the total number of movies released in that specific category. For example, if we were interested in comparing the number of PG movies to PG-13 movies in our data set, we can quickly compare the length of the two bars representing the number of observations. We can display bar charts either vertically as shown or horizontally where the length of each bar now represents the number of movies released for each specific category. There are situations where one orientation may be more appropriate than the other, but in general both horizontal and vertical bar charts communicate the same information. In addition to comparing the frequency of observations across categories, we might also want to know how the number of observations relates to all observations, that is, parts of the whole. We can visualize parts of the whole as a pie chart. In this pie chart we have the proportion of movies released in each rating category. The interior angle of each wedge represents the proportion of movies released in the specified category, however, pie charts are generally discouraged by data visualization experts because the human brain isn't as good at comparing values using the angles in a pie chart relative to using the lengths of bars in a bar chart. This is especially the case when there are a large number of categories in the pie chart. However, if you keep the number of categories limited to a very small number, for example two categories, and your visualization isn't intended or a precise comparison, there are places where pie charts may be acceptable. Please use your best judgment and if all else fails, just use a bar chart. There are other data visualizations we can create for qualitative univariate analysis, but these should cover the basis. The second type of visual analysis we can perform is quantitative univariate analysis, that is the visual analysis of a single numeric variable. When we're performing univariate analysis of quantitative variables, we're typically concerned with the location, spread, and the shape of the data for a single numeric variable. We'll take a look at visualizations that capture each of these features next. For our quantitative univariate analysis visualizations, we'll once again use a randomly generated sample of values produced by the normal distribution. This is a distribution we used in the previous module that models the probabilistic outcome of many of the processes and phenomena in our world. First, if we're interested in seeing a visual representation of the observations of our variable, we can create a dot plot of the observations. A dot plot creates a point represented by a circle in this case for each observation along the dimension of our variable. This dimension could be the quantity of pizza sold, movie runtime, box office revenue, et cetera. However, in this example, we just randomly generated values using the standard normal distribution, that is a normal distribution with our mean set to 0 and our standard deviation set to 1. On the X axis we have the possible values of our normally distributed variable ranging from app negative 4 to positive 4. Each circle represents an individual observation at a specific value along our X axis. We view circles because it's easier to see where they overlap and thus where clusters of overlapping observations are. With a dot plot we can quickly see how the data are distributed along the dimension of the variable. Next we can represent the same data as a box plot. A box plot, also known as a box and whiskers plot, provides us with a visual representation of the five number summary we learned about in the previous module. In addition, the box plot also provides us with information about potential outliers, that is values that fall outside of the statistically likely minimum and maximum values entailed in a distribution of values. Outliers may be naturally occurring or the result of measurement error and thus sometimes they may need to be excluded from our analysis. On the X axis, we have the possible values of our normally distributed variable. As we can see, the visualization provides us with the minimum value observed for the variable, the lower or first quartile, the median, the upper or third quartile, the maximum value observed for the variable excluding any outliers, and any outliers in the data set represented as small circles which are values greater than 1. 5 times the upper quartile or less than 1. 5 times the lower quartile. A box plot allows us to quickly inspect the location and spread of our data. Next we can represent these same data as a histogram. A histogram shows us an approximation of the shape of the distribution of values by grouping them into equal width partitions called bins. On the X axis we have the possible values of our normally distributed variable. On the Y axis we have the number of observations. The height of the bar represents the number of observations contained within a specific bin. The width of all bins are all equal. In other words, each bar shows us the number of values that occurred between the minimum value of the bin and the maximum value of the bin. We can change the size of the bins to create either a more coarse-grained representation of the distribution or a more fine-grained representation of the distribution. Histograms are often confused with bar charts. However, while a bar chart is showing a measure across multiple categories, a histogram is showing the frequency of observations of a numeric variable grouped into bins. Finally, we can represent these same data as a density plot. Density plots, more precisely called kernel density plots, show us the shape of a distribution of values similar to a histogram. On the X axis, we have the possible values of our normally distributed variable. On the y axis, we have the probability, which ranges from 0 to 1, of a value occurring at a specific location. Where a histogram shows us a step wise approximation of the shape of a distribution using bins, a density plot shows us a smooth representation of the distribution as a function. In addition, while the histogram showed us the number of observations in each bin, the density plot shows us a probability density function. A probability density function tells us the likelihood expressed as a number from 0 to 1 that an observation will be found at a specific point on the line. We can use the area under the curve to compute the likelihood that an observation will occur within a range of values. Thus, the entire area under the curve should integrate to 1. To explain how all of this works requires a bit of calculus, but we won't need to know any of that for the demos in our course. We're just going to be looking at the shape of our data, not calculating probabilities using a probability density function.

Bivariate Analysis

The third type of visual analysis we can perform is qualitative bivariate analysis, which is the visual analysis of two categorical variables. When we're performing qualitative bivariate analysis we're typically interested in the joint frequency of observations, that is the frequency of occurrences of observations at the intersection of two categories, for example, movie genres and movie ratings. We can visualize bivariate frequency distributions using a spine plot, which is a series of stacked bar graphs whose size corresponds to the number of observations at the intersection of each category. On the X axis we have movie genres; on the Y axis we have movie ratings. The width of the bars represents the proportions of observations in each of the categories on the X axis. The length of each segment in a stacked bar represents the proportion of observations in each category on the Y axis. The color corresponds to the category on the Y axis. Color is used to help visually distinguish categories since they may vertically overlap from one bar to the next. For example, in this spine plot, if we were interested in seeing how many PG rated comedies that are in our data set, we would find the stack bar at the intersection of the comedy column and the PG row. There's a variation on the spine plot called a mosaic plot where the spacing between the bars and segments produces a cleaner data visualization. It works the same as a spine plot in that the width of the tiles represents the proportion of observations on the X axis and the height of the tiles represent the proportion of observations on the Y axis, thus we can use the area of the tiles to determine the number of observations at the intersection of each of the two variables. Color is once again used to help distinguish categories on the Y axis since they may not line up vertically. Black dashes indicate that there are no observations at the intersection of the two categories. For example, there are no G rated crime or horror films, which should match our intuition. Once again, if we were interested to see how many PG rated comedies there are in our data set, we would find the tile at the intersection of the comedy column and the PG row. The fourth type of visual analysis we can perform is quantitative bivariate analysis, which is the visual analysis of two numeric variables. In quantitative bivariate analysis, we're typically looking at the relationship between two numeric variables. To visualize relationships between two numeric variables we typically use a scatterplot. We'll use our two variables from the Old Faithful data set to illustrate our scatterplot. On the X axis we have the duration of eruptions in minutes. On the Y axis we have the duration of time before the next eruption in minutes. Each point represents an observation of a geyser eruption. The location of each point corresponds to a specific duration of eruption given by its location on the X axis and duration before the next eruption given by its location on the Y axis. Since the points have a general slope from the X/Y original, that is the lower left hand corner, pointing to the upper right hand corner, there is a positive correlation between these two variables. If the slope of the data points were from the upper left hand corner to the lower right hand corner, we would say that the variables are negatively correlated. If the line were perfectly flat, we would say that there is no correlation between the two variables. Scatterplots allow us to quickly see patterns in our data in ways that would likely take much longer with numerical techniques. If one of our two quantitative variables is a unit of time, we can typically visualize these two variables using a line graph to see trends in the data over time. A sequence of data measured over time is typically referred to as a time series. On the X axis, we have the year a movie was released. On the Y axis, we have the total number of movies released that year. The height of each point connecting the lines corresponds to the number of movies released in a specific year. The lines connecting the points show us the rate of change from one year to the next. A line graph is similar to a scatterplot in the sense that each data point is displayed on a two-dimensional X/Y coordinate plane. However, unlike a scatterplot, each value on the X axis can have one and only one value on the Y axis. In addition, each value on the X axis should typically be uniformly spaced. Moreover, each point on the 2D surface is connected by a line to help visualize the rate of change between each point. Please note that while this graph appears to indicate that the number of movies being produced has decreased dramatically in the past 2 years, this is simply because the newer movies have not yet been recorded in our movies database. This is a perfect example of how understanding the context surrounding your data is critical to avoid making errors and incorrect claims about your data. The fifth and last type of analysis we will cover is bivariate analysis of both a qualitative and a quantitative variable, that is one categorical variable and one numeric variable. If we want to compare a single numeric measure across multiple categories, we would typically visualize this using a bar chart. For example, we could compare the average critic score for our movies by their rating category. On the X axis we have the rating categories, on the Y axis we have the critic score, which is measured from 0 to 100%. The height of each bar represents the average critic score in each rating category. For example, we can quickly see that critics rated PG movies much higher than R rated movies. If we are interested in seeing the 5 number summaries of a numeric variable across a set of categories, we can create multiple box plots for each category. This allows us to quickly compare the measures in the 5 number summaries, that is minimum, lower quartile, median, upper quartile, and maximum across each category. For example, we can quickly see that the median critic score of G rated movies is much higher than the median critic score of PG-13 movies. When we have a series of the same type of data visualization side by side on the same axis, this is typically referred to as small multiples, a trellis chart, or a lattice chart. There are many more types of data visualizations we could look at, including visualizing three or more variables at once. However, the visualizations that we've seen thus far, which are all produced using the basic plotting system in R, should get us started for our exploratory data analysis.

Guidance

Before we see how to create these various types of data visualizations in R, I think it's important that we spend a bit of time discussing how to avoid making errors and invalid claims with our data visualizations. This is a very important topic because it's very easy to unintentionally mislead others or misrepresent data when we create our data visualizations. First we must know our audience. We should always consider who will be viewing our data visualizations and then create our visualizations accordingly. We could producing our data visualizations for a small audience who is very familiar with the data and the context or we could be producing our data visualizations for a large audience who is unfamiliar with the data or the context. We should invest time and effort in proportion to the size of the audience, their familiarity with the data and domain, and based on the potential impact of decisions that will be made using our data visualizations. Data visualizations created for exploratory purposes are typically referred to as exploratory data visualizations. Whereas data visualizations created for a wide audience are typically referred as expository or explanatory data visualizations. If we are performing exploratory data analysis, we are typically creating our initial data visualizations for an audience of one, that is, we're creating the data visualizations for ourselves using our data exploration process. We typically do this in a very quick and dirty way because we're only concerned with learning from our data exploration, not communicating information to a wider audience. In addition, we typically have a deep understanding of the data and the context, so we don't need to clean up the data visualization or create human readable titles, labels, et cetera. The out of the box defaults are typically good enough for the data exploration process. However, if we're planning to create data visualizations to be shown to a wide audience, we would need to spend extra time and effort cleaning up the data visualizations, adding human readable titles, labels, legends, and more. Creating high quality expository data visualizations to be communicated to a wider audience, however, is outside of the scope of this course. Second, we should start each data visualization with a question that we're attempting to answer. For example, our question might be, which movie rating category had the most movies released from 2000 to 2015? Once we have our question in mind, we can then attempt to answer that question using a data visualization. In this case our question involves a comparison of observations for a qualitative variable, that is, movie ratings. So, we can create a bar graph of the number of movies by rating category to visualize the answer to our question. Starting with a question in mind helps us to focus on a specific goal and helps us choose a data visualization that is appropriate to answer the question. Third, it's important that we use the right tool for the job. This starts with knowing our audience and the question that we're trying to answer for our audience. Ask yourself, what information is the audience trying to learn from our visualization? Are they trying to compare two values to see which is larger? Are they trying to see rates of change over time? Or are they trying to see how values are distributed across a variable? For example, if our audience is interested in comparing movie rating categories, a pie chart may not be the best way to represent this information. However, a bar chart is an excellent tool to communicate this information for comparison purposes. Once we know the question our audience is asking, then we need to know which type of data visualization communicates the answer in the most effective way for our audience. This requires learning which types of data visualizations are most appropriate to answer specific types of questions in specific contexts. Fourth, we want our data visualizations to be clean and simple. Our goal is to maximize the data to ink ratio of our data visualizations. We want to communicate as much information as possible using the minimum amount of digital ink. This means that we should strive to eliminate all chart junk as data visualization expert, Edward Tufte, likes to call it. That is, anything on our chart that does not directly help communicate the core information of our data visualization. For example, here's a data visualization that contains a tremendous amount of chart junk. It is a 3D graph, but the third dimension communicates no relevant information. It has each bar color coded and a legend, but this is redundant information since each bar has an X axis label. Each bar also has unnecessary special effects like rounded corners, surface reflectivity, and drop shadows. It has a beige texture on the main canvas and the back wall of the bar graph has a granite texture. It uses multiple fonts that are difficult to read, are heavily bolded, and use 100% black text. In addition, we also has thick, solid, Y axis grid lines that add even more clutter to this already heavily cluttered graph. Instead, we can communicate the exact same information with a much simpler graph like so. We've eliminated the unnecessary 3D bars, special effects, and backgrounds. We've eliminated the redundant color coding of the categories and the legend, we've cleaned up the fonts, removed the bolding, and lightened the text, and we thinned out and lightened up our Y axis gridlines. When we're creating data visualizations we want to keep things clean and simple. Oftentimes less is more. Finally, we want to avoid information distortions in our data visualizations. It is very easy to lie with data, whether these lies are intentional or unintentional. In fact, there are so many ways to visually misrepresent data using data visualizations, that entire books have been written on the subject. For example, let's say we're interested in comparing the total number of PG-13 movies to R rated movies. We could create a graph where the Y axis begins at an arbitrary location, for example 1200 movies. This makes it appear as if there's a significantly larger number of R rated movies than PG-13 movies. However, if we create this graph with the Y axis starting at 0, these two values don't look all that different after all. Simply by changing the start and end of our Y axis we can unintentionally distort the reality of our data. Other examples of information distortion involve using exaggerated 3D perspectives, using the size or shapes rather than their area for comparison, and breaking established data visualization conventions. It is important that we learn how someone can lie with data visualizations in order to avoid accidentally lying with our own data visualizations. In addition, it's also a very important skill for recognizing when we're being lied to by someone else's data visualization.

Demo Setup

The next day our friend, coworker, and CFO, Joe tells us that he's now convinced that his movie idea isn't likely to be a box office hit after all. And the type of movie our data analysis suggested, that is a PG rated, 100 minute fantasy, action, or adventure movie, is actually a much better idea to pitch to the board of directors. So Joe stayed up all night and came up with a new idea for a movie that he thinks will be a box office smash. It's an action, fantasy, adventure, that's PG rated and 100 minutes long. He still thinks critics will hate it, however, he doesn't care what the critics think. He's convinced this thing's going to be a big hit at the box office. In fact, he's perfectly fine creating a movie that critics will hate because he believes that critics only like stuffy art films, not real movies like the movie he's proposing. In addition, we already showed him that critic score doesn't strongly correlate with box office revenue anyways. He wants to pitch his movie idea to the movie studio's board of directors at the meeting this afternoon. However, he says that he's not really much of a numbers kind of guy, keep in mind he's our chief financial officer, but if we could help him present the same information we showed him at the bar in a more visual way he thinks he could convince the board of directors to green light his new movie. So we're going to help Joe once again by performing an exploratory data analysis in a visual way. First, let's start again by setting our working directory to the folder containing our source data file. Next, we'll load our csv file, movies. csv using the read. csv command. In addition, we'll also load our genres data file as well. Let's also take a peek at the movies data set again using the head command. We can see that we have all the columns and data that we would expect. We can also take a peek at our genres data set too and we see that we have all of the columns and rows that we'd expect as well.

Univariate Analysis Demo

The first type of visual analysis we'll perform is univariate analysis of a qualitative variable, that is the visual analysis of a single categorical variable. We'll do this by using the plot command on a categorical variable like movie ratings. As we can see, this returns a bar graph of the frequency of occurrences of movies in each rating category. For example, we can see that there are very few G rated movies, more PG movies, relatively more PG-13 movies, and the mode that is the category with the most observations is the R rated movie category. If we're interested in visualizing a qualitative variable as a part of the whole, we can create a pie chart using the pie command and pass a frequency table of the movie ratings variable in as an argument. In this case, it produces a pie chart with each wedge and color assigned to a specific movie rating category. As we can see again, the number of observations increased from G to PG, to PG-13, and finally R movies. Keep in mind that most data visualization experts discourage the use of pie charts for visual comparison because the human brain isn't as good at comparing angles as it is to comparing the heights or lengths of bars. The second type of visual analysis we'll perform is univariate analysis of a quantitative variable. That is, the visual analysis of a single numeric variable. First we'll create a dot plot to see the actual location of each observation along the dimension of the quantitative variable. To create a dot plot of movie runtimes we'll use the plot command. Set the x coordinate of the points in our plot to the movie runtime variable, set the y coordinate of the points in our plot to 0 for all values so that they are on a straight line using the replicate command repeating the value 0 once for each row in the movie runtime variable. We'll also set the Y axis title to an empty string since there's no dimension assigned to the Y axis and we'll set the Y axis type to n for none to suppress the plotting of the Y axis labels and tick marks. On the X axis, we have the movie runtimes in minutes. We have no dimension on the Y axis, since we're only analyzing a single quantitative variable. As we can see, this produces a dot plot of each movie's runtime. In addition, we can see the minimum value, maximum value, and a large cluster of observations in the center of the distribution of values. Next, we'll create a box plot of the same data. We'll do this using the boxplot command, then we'll set the x coordinate to the movie runtime variable, we'll set our x axis label to runtime in minutes since there's no label for this plot by default, and we'll specify that the orientation of our box plot should be horizontal to mirror our previous dot plot. On the X axis we have our movie runtime in minutes. Once again, we have no dimension on the Y axis since we're only looking at a single variable. As we can see, this produces our 5 number summary along with small circles for outliers, that is observations greater than 1. 5 times the upper quartile or less than 1. 5 times the lower quartile. We have our minimum value in our data set, outliers on the lower end of the variable, minimum value excluding outliers, lower quartile cutoff point, median, upper quartile cutoff point, maximum value excluding outliers, outliers on the upper end of the variable, and maximum value in our data set. Graphically we can see that our movie runtimes center around 100 minutes with a longer tail on the right side of the distribution. Next, we'll create a histogram of the movie runtimes. We'll do this using the histogram command, passing in our movie runtime variable as an argument. On the X axis we have movie runtime in minutes. Notice that the plot rendered the variable name as our default X axis label. On the Y axis we have the frequency, that is, number of observations. Each bar represents a bin in the histogram and the height represents the number of observations in each bin. As we can see, this creates a histogram of the distribution of movie runtimes. Once again, the shape matches what we've seen previously, that is, values center around 100 minutes and the distribution is positive skewed, that is, the tail on the right side is longer. By default the plot rendered with 20 bins, however we can create a more coarse-grained histogram by setting the number of bins to 10 or we can create a more fine-grained histogram by setting the number of bins to 30. Notice that this is still the same data and distribution, we're just changing the number of bins that we're using to approximate the shape of the distribution. If we were to continue increasing the number of bins in our histogram and we have a sufficient number of observations in our variable, the shape of the histogram would continue to become smoother and more closely approximate the actual shape of the distribution. However, if we want to estimate the shape of the distribution as a smooth curve, we can create a kernel density plot of the distribution using the density command on the runtime variable and pass the result into the plot command. On the X axis we have movie runtime in minutes. On the Y axis we have the probability density, that is, relative likelihood from 0 to 1 that an observation will be found at the corresponding location on the X axis for this variable. The shape of the curve approximates the underlying distribution of values and the area under the curve integrates to 1, that is, 100% of the values can be found underneath the curve. Just to tie this back to the original dot plot we created for the movie runtime variable. We'll plot these same points at the base of this density plot. We'll do this using the points command to add our observations to our plot. We'll set the x coordinate to the movie runtime variable and we'll set the Y axis using the replicate command again, setting each value to 0. 0005 so that it will show up right blow the Y axis origin, that is the horizontal line that runs the length of the chart at a probability density of 0. As you can see, the curve of this density plot does in fact match the observations in our data set.

Bivariate Analysis Demo

The third type of visual analysis we will perform is bivariate analysis of qualitative variables. That is, the analysis of two categorical variables. To perform this type of analysis, we will create a spine plot of two categorical variables, that is, movie genre and movie ratings. To create a spine plot we use the spine plot command and set our X axis to the movie genre variable and our Y axis to the movie rating variable. On the X axis we have movie genres, on the Y axis we have stacked bars of movie ratings. The width of each bar corresponds to the proportion of movies contained in each genre. The height of each colored bar segment corresponds to the proportion of movies contained in each specific rating category for that corresponding movie genre. The colors are used to visually identify the rating category since the bar segments might overlap vertically. The X axis title and the Y axis title are set to the names of their respective variables. In addition, we can see that some of the X axis labels are missing because they can't all fit in the graph horizontally. We could fix this by expanding the diagram, filtering out genres, or rotating the labels if we wanted, but since this is just an exploratory data analysis, we'll just leave it as is. As we can see from our visualization, there are a lot of R rated dramas. However, there are very few G rated action movies. We'll visualize these two qualitative variables in a slightly different way using a mosaic plot. We'll also rotate our X axis labels to make more room for our genre labels. To do this we'll use the mosaicplot command and we'll set the x argument to a contingency table containing both movie ratings and movie genres. There is no y argument for a mosaic plot, which is a bit inconsistent from the other plotting commands we've seen. In addition, we'll rotate the X axis labels perpendicular to the X axis to make more room for the genre labels by setting the label axis style to always vertical using the argument las equal to 3. When setting label access style with the las parameter, las equal to 0, which is the default, means that the axis labels are always parallel to the axis. Las equal to 1 means that the axis labels are always horizontal. Las equal to 2 means that the labels are always perpendicular to the axis, and las equal to 3, like we have set, means that the labels are always vertical. On the X axis we have our genres. On the Y axis we have our ratings, however their order is reversed from that of our spine plot. The width of the tiles is proportional to the number of movies in each genre. The height of the tiles is proportional to the number of movies in each rating category for the specific genre. And the small dashed lines are used to indicate tiles where there are no observations, thus the area of each tile is directly proportional to the total number of observations at the intersection of each genre and each rating category. Unlike most plots in R, the mosaic plot places the X axis labels at the top of the chart instead of the bottom. In addition, notice how the labels on both the X axis and the Y axis are vertical, as specified by our las equal to 3 parameter. Unfortunately when we rotate the Y axis labels to be vertical, the label text is right justified rather than left justified, so our labels look a bit funny. We could fix this with more advanced plotting techniques or by using a different mosaic extension package in R. However, for our exploratory data analysis, this should be good enough. Notice again that it's very easy to see that there are a large number of R rated dramas, however, there is a much smaller number of G rated action movies. The next type of visual analysis we will perform is bivariate analysis of quantitative variables, that is the visual analysis of two numeric variables. To perform this type of analysis we will create a scatter plot. We'll start by creating a scatter plot of movie runtimes and box office revenue. To do this we'll use the plot command. We set the x coordinate of the plots to the runtime variable and set the y coordinate of the plots to the box office revenue variable. On the X axis we have movie runtime in minutes. On the Y axis we have box office revenue in millions of dollars. The X and Y axis titles are set to their two respective variables. Each circle in the scatter plot represents a movie located at the corresponding movie runtime and box office revenue. As we can see, there's a gradual increase in the box office revenue as movie runtime increases. So the two variables are positively correlated. However, it does not appear visually to be a very strong correlation. In addition, we'll also create a scatter plot of movie critic score and box office revenue. As we can see, we have a positive correlation between critic score and box office revenue. However, it is also a relatively weak correlation. So unfortunately neither movie runtime nor critic score would be useful as a predictor for our box office revenue. In addition, it is still the case that correlation does not imply causation. So in the case of Joe's new movie, he should probably just stick with a 100 minute runtime since that's what the average movie runtime is and we have no evidence that increasing or decreasing movie runtimes would have any significant effect on box office revenue. In addition, whether critics like a film or not doesn't seem to have a significant impact on box office revenue either. So we have no evidence to say that Joe creating a movie that critics will hate might have a negative or positive impact on box office revenue. In fact, I'm sure we can all think of several movies that critics hated, but did extremely well at the box office. Next, we'll create a line graph of the number of movies released each year. Unfortunately our data set isn't structured as a time series, so we're going to have to improvise a bit. We're just going to count all the movies that were released each year and use year as our temporal variable and the count of the movies as our second numeric variable that is changing over time. So in essence, we're projecting a subset of our movies data set into a new data set with two numeric variables, that is year and count of movies released each year. To plot this line chart we use the plot command. We set x to our frequency table of movie counts by year, this sets our x coordinate to the year a movie was released, our y coordinate will be set to the number of movies released that year. We also set the type of the plot to l for line, this tells the plot command to create a line graph. On the X axis we have the year, on the Y axis we have the number of movies released. The points represent the number of movies released on a specific year and the lines represent the rate of change from one year to the next. We can see that the number of movies released each year has continued to increase from 2000 until about 2013, at which point the graph sharply drops off. However, this sharp drop off is only because many of the newer movies have not yet been added to our movies data set. The last type of visual analysis that we will perform is bivariate analysis of both a qualitative and a quantitative variable, that is the visual analysis of one categorical and one numeric variable. First we start by visualizing the average box office revenue of each rating category. We'll do this by using the tapply command like we did in the descriptive statistics demo. However, we'll use the output of our tapply command as the height argument for our bar graph. On the X axis we have movie rating categories, on the Y axis we have box office revenue in millions of dollars. The height of each bar corresponds to the average box office revenue in each movie rating category. As we can see, the R rated movies did not do very well on average, however, the G, PG, and PG-13 movies all performed about the same with PG movies performing slightly better than the other two categories. We'll also visualize the average box office revenue across each movie genre. We'll do this with the same commands, but we'll also modify the label axis style to render all of the axis labels vertically so we can read our genre labels easier. We do this by setting our label axis style equal to 3 again. On the X axis we have our movie genres, on the Y axis we have box office revenue in millions of dollars. The height of each bar represents the average box office revenue for each genre. As we can see, adventure, animation, fantasy, sci-fi, and action movies all perform very well at the box office relative to lower performing genres like westerns, musicals, and documentaries. So this data visualization reinforces Joe's desire to make a PG rated film that has elements of action, fantasy, and adventure. Finally, we'll visualize our 5 number summary of a quantitative variable across multiple categories. We'll do this using the plot command. We set the X axis to our movie rating variable and set the Y axis to our box office revenue. On the X axis we have our movie rating categories, on the Y axis we have our box office revenue in millions of dollars. In the plot area we have our box plots representing the 5 number summaries for box office revenue in each rating category. As we can see, we have a large number of outliers in the PG-13 category. We have a higher maximum value, excluding outliers, in the PG category. The upper quartile of the PG category is also higher. And the median values are about the same across the first three categories. Ultimately this further reinforces Joe's desire to recommend a PG movie to the board of directors.

Demo Conclusion

To keep things manageable for this course, we only attempted to visualize at most two variables. However, there are numerous ways to visualize three or more variables at once. While we won't be demonstrating how to visualize three or more variables in this course, I think it's important to show you how to quickly visualize an entire table of data at once. We'll do this using a scatter plot matrix. To create a scatter plot matrix we'll simply use the plot command and pass a data frame in as an argument. On the X axis we have each of the variables in the data frame. On the Y axis we also have each of the variables in the data frame. We have the variable names on the main diagonal of the matrix. Then in each cell of the matrix we have a small multiple scatter plot of the two intersecting variables with the variable on the x column as the x axis of the scatter plot and the variable on the y row as the y axis of the scatter plot. For example, if we're interested in seeing our scatter plot of movie runtime and box office revenue, we simply find the runtime column on the X axis and the box office revenue row on the Y axis and find the scatter plot at the intersection of those two variables. As we can see, we have runtime on the X axis of the scatter plot and box office revenue on the Y axis of the scatter plot. You should recognize this as the scatter plot we looked at earlier with the small positive correlation between the two variables. We'll do this once again just to make sure we understand how this works. For example, if we want to see our scatter plot of critic score and box office revenue, we simply find the critical score column on the X axis and the box office row on the Y axis. As we can see here, we have the same scatter plot we looked at earlier with the critic score on the X axis and box office revenue on the Y axis. In general, I don't recommend attempting to visualize more than three variables simultaneously, however there are cases where doing so, like a scatter plot matrix, makes sense. Also, don't worry if you're not immediately comfortable using a scatter plot matrix, it actually took me quite a bit of time to get used to working with them myself. We're not going to be creating any expository data visualizations in this course, that is production quality data visualizations to be communicated to a wider audience. However, before we conclude our data visualization demo, I'd like to briefly show you how you can clean up your data visualizations. In addition, you'll get a glimpse of the power and flexibility of R's plotting system. Joe wants to present several of these data visualizations that we've created to the board of directors. However, they're not currently in a state of consumption for a wider audience, so we're going to have to provide a bit more context of the data visualizations and clean them up a bit. For example, let's take a look at the first data visualization we created in this demo, that is count of movies by rating category. First we'll display this bar graph using the out of the box defaults. While this visualization is fine for us because we know the data and context well, a third party would have a very difficult time making sense of this data visualization. Luckily for us, R does a pretty good job with its out of the box defaults for creating relatively clean data visualizations, so we really don't need to do much clean up here. However, to provide a bit of context to this data visualization and to make it presentable to a wider audience, we'll add a few basic elements to our plot. We'll add a main title, we'll add an X axis title, a Y axis title, and we'll specify a primary color for our data visualization. As we can see, this added the main title, access titles, and color to our bar graph. But this is just the tip of the iceberg for customizing data visualizations. There are so many ways to enhance data visualizations in R that it would take an entire course to cover them all. In fact, we can even create our own data visualizations piece by piece by creating each of the elements of the data visualization from the ground up. Just to get you pointed in the right direction, be sure to take a look at the help files for the plot command, as well as the parameters for the plotting system. The help files will show you all of the graphical parameters you can modify. Joe presents his movie idea and our data visualization to the board of directors. They are very impressed with Joe's creativity and apparent depth of knowledge about the movie industry. They green light Joe's film and production soon begins on Joe's new blockbuster smash hit. Once again, keep in mind that this narrative is all in fun, just to keep things a bit more interesting. We really should not be attempting to predict whether a movie will do well at the box office based upon the limited data and limited analysis techniques that we have at our disposal in this course. It's fine to use our data set to describe what happened in the past, but it would not be acceptable to attempt to make predictions about box office revenue of movies in this manner.

Summary

In this module we learned about data visualization and how it can be used to represent our data in visual ways. Next, we learned about the various types of data visualizations that we can create to visualize our data. Then we learned how to avoid making errors with our data visualizations. Finally, we saw a demo that tied all of these concepts together. In the next module we'll learn about several concepts that go beyond R and exploratory data analysis.

Moving Beyond R and Exploratory Data Analysis

Overview

Hello and welcome back to this final module in Exploratory Data Analysis with R. I'm Matthew Renze with Pluralsight and in this module we'll look at a few topics that move beyond R and exploratory data analysis. First, we'll start with other types of data analysis we can perform beyond exploratory data analysis. Next, we'll see a demo of two additional types of analysis we can perform, that is linear regression analysis and cluster analysis. Then we'll learn about alternatives to R for performing exploratory data analysis. Finally, we'll wrap things up by concluding the module and the course as a whole.

Beyond Exploratory Data Analysis

I really want to emphasize that what I've shown you in this course so far is just the tip of the iceberg. I can't stress how much more R provides beyond what we've seen in terms of data analysis, data visualization, and programming features. First, there are several other major categories of data analysis beyond what we saw in this course. We mostly looked at descriptive statistical analysis, that is, describing the features of our data in meaningful ways, and exploratory data analysis, the topic of this course. Descriptive statistical analysis is generally the least complex and difficult to perform. Exploratory data analysis is generally next in terms of difficulty or complexity. However, there are four other main categories of data analysis beyond the two we've touched upon in this course, which we'll briefly discuss in order of relative difficulty or complexity. Third, we have inferential data analysis, that is testing a hypothesis about the world by collecting a sample of data and generalizing it to a larger population. This is the type of data analysis that is taught in introductory college statistics courses. If you've taken one of these courses, you'll probably recognize terms like null hypothesis and P values. These are terms frequently used in inferential data analysis. Fourth, we have predictive data analysis, that is using current or historical data to make predictions about future or otherwise unknown values. This type of data analysis is typically taught in higher level college statistics and machine learning courses. It is rapidly growing in popularity in the business world where it is commonly referred to as predictive analytics. Fifth, we have causal data analysis, that is determining how modifying one variable while holding all other variables constant changes some outcome variable. This type of data analysis is typically taught in graduate level college statistics and scientific research courses. It is commonly found in clinical research, for example, using randomized, double blind, placebo controlled studies to determine the efficacy of pharmaceuticals on certain human diseases. Finally, we have mechanistic data analysis, that is creating a mathematical model that captures the exact changes in all variables as one variable is modified. This type of analysis is typically taught in advanced college courses in engineering and physical sciences. It is only possible in systems that have variables that can be modeled as deterministic, that is, non-probabilistic sets of equations. For example, an engineer might create a mechanistic model of an automobile accelerating when the gas pedal is pressed using data from various sensors in the automobile. R has commands, algorithms, and extension packages that can assist you in performing all six of these types of data analysis. I should note, however, that while there are various ways to divide the space of data analysis into multiple categories, I tend to prefer using these six categories as identified by Jeff Leak of Johns Hopkins University. This course only provided a brief introduction to the programming features of R, just enough to get us started with our exploratory data analysis. However, R is a feature rich, multi-paradigm language supporting aspects of procedural programming, functional programming, and object-oriented programming. In addition, R supports mixed programming allowing you to make outbound function calls to code written in C, C++ or Fortran, or you can make inbound calls into R from other programming languages like C#, Java, and Python. In addition, R can be used for distributed programming to handle big data scenarios using third party distributions like Revolution R Open from Revolution Analytics. Beyond the basic types of statistical analysis we saw in this course, R can do much more advanced forms of statistical analysis, some of these more advanced statistical analysis methods include statistical modeling where we model the relationship between variables as a mathematical equation, cluster analysis where we organize a set of data into groups that have similar properties, dimensionality reduction like principal components analysis and singular value decomposition where we reduce the data to a set of vectors that represent the variance of the data, Analysis of Variance also known as ANOVA, and many more. We'll perform basic statistical modeling, specifically linear regression and cluster analysis in our upcoming demos. In addition to numerical analysis, R can create significantly more advanced data visualizations than the out of the box data visualizations we saw in our demos. For example, the following data visualizations were all produced using R with more advanced data visualization packages. We can create tree maps, choropleths, bubble charts, network visualizations, stacked line graphs, even Chernoff faces. R also has many other data visualization packages like ggplot2, lattice, and hexbin that allow you to create much more advanced data visualizations. In addition, we can also create interactive data visualizations with R using extension packages like iplots and host our visualizations on the web using R Studio's web application framework called Shiny. Once again, this is just scratching the surface of the data visualization power of R. As I mentioned earlier in this course, data visualization relies on human perception to recognize patterns in data. While this is quite powerful, it doesn't scale well into higher dimensional data sets, that is data sets with a large number of variables. So if we have to work with high dimensionality data sets, we need machine assisted tools for pattern recognition. Two areas of data analysis that have these more powerful tools are data mining and machine learning. R has very powerful extension packages for performing data mining and machine learning. Unfortunately an in-depth discussion of data mining and machine learning is beyond the scope of this course, however, in our demo we will get to see a glimpse of linear regression analysis and cluster analysis, which are two commonly used tools in data mining and machine learning. I should note that these two images are not from R packages, but rather from an open source machine learning tool kit that I created in college to help people understand how machine learning algorithms work by visualizing the algorithms. Now let's switch gears and take a quick look at a demo of a couple more advanced types of analysis, that is linear regression analysis and cluster analysis.

Linear Regression Analysis Demo

One night as we're having drinks at the bar after work, Joe tells us that he's lost his passion for creating cheap knock offs of Hollywood blockbuster movies. He tells us that his real passion in life has always been plants, specifically flowers. He's been thinking about getting out of the movie business to open his own high end flower shop to cater to the stars in Hollywood. Apparently through his Hollywood connections he's discovered that there's a huge market for selling flowers that match the buyers exact specifications. They use them to create very expensive artwork and elaborate floral decorations so the flower petals need to be roughly all the same size and shape to be aesthetically pleasing. However, they are willing to pay top dollar for someone who can provide batches of flowers with all the correct dimensions and characteristics. To pull this off Joe has come up with an ingenious way to quickly measure the length of the petals of a specific type of flower called an iris using an optical scanner. However, this method doesn't work well for the widths of the petals since they tend to bend more and thus can't be measured accurately by the optical scanner. But he's noticed that the length and the width of the petals look like they may be related to one another. So he's collected precise measurements of 50 randomly selected flowers from three different types of iris flowers for a total of 150 flowers. His data set contains the petal length, that is the average length of the petals of the iris flower in centimeters; petal width, that is the average width of the petals of the flower; sepal length, that is the average length of the sepals, which are the leaf-like structures that enclose the petals until the flower blooms; sepal width, that is the average width of the sepals; and species, of which there are three species of iris flowers he's selling, Iris virginica, Iris versicolor, and Iris setosa. He asks us if we could analyze the data he collected to see if there might be a way to predict the width of the pedals given one of the other measures in the data set. So we offer to take a look at Joe's data set to see what we can do for him. First let's start by loading the data. The iris data set, which is a very well known data set for teaching data analysis and machine learning, is one of the many sample data sets that's included with R when you install it. So to load it we just use the data command and pass iris in as an argument. Next, we'll take a quick peek at the data using the head command. As we can see, we have five columns contained in the data set, sepal length in centimeters, sepal width, petal length, petal width, and species. We can see how many unique species that are using the unique command on the species column. As we can see, we have three unique species, setosa, versicolor, and virginica. Now let's create a quick scatter plot matrix of the first four columns of our data set to get a quick feel for the relationships that exist between the variables in the data set. As we can see there appears to be some patterns that exist within our data set. Let's take a look at the relationship between the petal length variable on the X axis and the petal width variable on the Y axis. It looks like these data have a relatively clear pattern to them. So let's zoom in on the cell in the scatter plot matrix by creating a scatter plot of those two variables. On the X axis we have petal length, on the Y axis we have petal width. As we can see, we have what appears to be a linear relationship between the two variables, that is the points generally follow a straight line. In addition, the line points to the upper right hand corner, which means that there is a positive correlation between these two variables. Moreover, the points are all relatively close together on that imaginary straight line, which means we probably have a relatively strong correlation between the two variables. If the points were more spread out, that would suggest a weaker correlation between the two variables. Also, the variability of the petal width as the petal length increases appears relatively stable, that is the variability of the points on the Y axis doesn't change very much as we move further along the X axis. This property is referred to as homoscedasticity whereas if the variability changed along the line it would be referred to as heteroscedastic. The fact that this is a linear relationship, the correlation appears strong, and the data are generally homoscedastic, are all good indications that we might be able to create a linear function that we can use to estimate the petal width given a petal length. Now let's try to create a linear regression model to see if we can predict petal width given a known petal length. To keep the variable name simple, we'll just map petal length to a variable called x, that is our predictor variable and we'll map petal width to a variable called y, that is our outcome or response variable. We can create a linear regression model using the lm command, which stands for linear model. We pass in a model formula written in the form outcome variable, the tilde character, followed by the predictor variable, which in this case we can read as petal width as a linear function of petal length. Then we'll set this linear model equal to a variable that we'll call model. Please note that the variable names that we use in our model formula are important. We'll see why this is shortly. To see how this linear model fits the data points, we can draw this linear function on the surface of our scatter plot. We do this with the lines command setting x equal to our petal length variable, y equal to our models fitted values, we'll set the color to red and we'll set the line width equal to 3. As we can see, this draws a linear function which represents our linear model on the surface of our scatter plot. Based on a visual inspect this line appears to be a good fit for the data, that is the data points are all generally centered around the line and there doesn't appear to be any outliers. Given how close all of the points are to the line, based on our visual inspection, this appears to be a relatively strong correlation. So let's double check the correlation via numerical analysis to see what the correlation coefficient is for these two variables. As we can see, we have a correlation coefficient of approximately 0. 96. This is a very high correlation coefficient and thus it appears that petal length may be a good predictor variable for petal width. Once again, correlation does not imply causation. We're not saying that increases in petal length are causing increases in petal width, rather we're only stating that increases in petal length are associated with increases in petal width. With our linear model and correlation coefficient in hand let's take a look at how we could use this model to predict new values for petal width given petal length. We'll do this by executing the summary command on the linear model. Summary gives us a lot of information about the linear model, most of which we would use if we were actually performing linear regression analysis. However, since we're just getting a taste for statistical modeling, we'll just going to focus on the key information. In the estimate section, we have two estimates, first we have our y intercept estimate, which is approximately negative 0. 36, this is the value on the Y axis where the regression line would intersect with the Y axis if we drew the regression line all the way to the Y axis line. Second, we have the estimate for the slope of the line, this values tells us that for each 1 centimeter increase in petal length, the petal width increases by approximately 0. 4 centimeters. Using just these two values we can draw the line that we see on the scatter plot. In addition, using only these two values we can predict, with a given degree of accuracy, what the petal length of a flower would be given the petal width. Simply multiply the length of the petal by the slope of the line and add the Y intercept value. For example, if we have a flower with a petal length of 2 centimeters, we can calculate the predicted petal width as 2 times 0. 41 plus -0. 36, which is equal to approximately 0. 46. As we can see, this value matches our visual prediction using our linear regression line. If we want to predict a large number of values, instead of calculating the values by hand we can use the predict command.

First we set the object argument equal to our linear model, then we set our new data argument equal to a data frame containing a column which each of our predictor values, that is our petal length. We'll just add three predictor values, that is petal lengths of 2, 5, and 7. As we can see, the 3 predicted petal widths are approximately 0. 46, 1. 71, and 2. 54. All three of these predicted values should match what our linear regression line visually predicts. Please note that the name of the predictor variable in our data frame must match the name of the predictor variable we used in our formula. I've made this mistake several times now, not having the variable names match up, and it's not immediately obvious what went wrong or how to fix the issue when it happens, so just keep an eye out for it. Since we're just trying to get a sense of the other types of analysis that we can perform in R, I've skipped over a lot of very important information that would typically be covered in a statistics course dealing with regression analysis. There are many assumptions that must be made about our data and many tests that we'd want to perform in order to determine whether a specific linear model is appropriate for our data. In addition, we typically want to provide our prediction with confidence intervals to indicate how precise our predictions are expected to be. However, these tasks are better explained in the context of a full course on statistical modeling or predictive statistics. If you find this type of data analysis interesting though, I highly recommend you seek out one or more courses that cover regression analysis in depth to learn how to perform this type of analysis correctly and to avoid making errors or invalid predictions with your statistical models.

Cluster Analysis Demo

Joe is very impressed with our linear regression analysis and he believes that he can use the linear model that we created to successfully predict within a certain degree of accuracy what the petal width will be given each flower's length. He's so impressed, in fact, that he proposes a much more difficult challenge to us. He says that also needs to organize batches of flowers by their species. He can inspect each flower by hand, but it takes too much time to be cost effective. He wonders if it might be possible to predict the correct species of iris flower given just the lengths and widths of the petals and sepals derived from his optical scanning technology. The three species of iris flowers that we have our Iris virginica, Iris versicolor, and Iris setosa. We think about the question for a minute and it occurs to us that we might be able to perform a cluster analysis to see if the data are separable into distinct and predictable categories for each species. However, we'll have to inspect the data again to see if this will be possible. So we're going to attempt to perform a cluster analysis using the K-means clustering algorithm to see if we can correctly classify the species of flower based solely on the lengths and widths of the petals and sepals of

Joe's flowers. One last time, let's load the data using the data command. We'll take a quick peek at the data using the head command and we'll look at each unique species of flower contained in the data set. Next we'll create a scatter plot matrix of the four numeric variables in the data set. However, this time we'll plot each species using a different color so we can easily spot clusters in the data set along each combination of variables. We'll do this by converting the species factor variable into its underlying integer representation and mapping that integer to an index of a color in the default color pallet. As we can see from the scatter plot matrix, there are definitely visually identifiable clusters of flower species in the data set. In addition, some of these clusters appear to be linearly separable, that is we could draw a line through the scatter plot in such a way that separates one colored set of points from all other colored sets of points. So far, things are looking pretty good for our cluster analysis. Before we create the clusters for our data, let's zoom in on a single cell of our scatter plot matrix. Let's look at the scatter plot with the petal length on the X axis and petal width on the Y axis. In addition, we can color the points in the scatter plot by species like we did in the scatter plot matrix to see how they are distributed. As we can see, in this 2-dimensional projection of our data set, all the setosa flowers in black are completely linearly separable from all other flowers. However, versicolor in red and virginica in green are almost but not quite linearly separable. That is, if we try to draw a line to separate them, some of the other species of flower will be on the opposite side of the line. Keep in mind that even though we're only going to visualizing our cluster analysis through a single 2-dimensional projection of our data set, our cluster analysis will be clustering data in a 4-dimensional space, that is across all 4 variables of our data set. To create our clusters using the K-means cluster analysis algorithm, we'll set a variable that we'll call clusters equal to the K-means comnad with the x argument set to the 4 numeric dimensions of our iris data set, set the centers arguments to 3 since we know in advance that we have 3 and only 3 types of species of flowers in the data set, and the number of algorithm restarts to 10. This tells the algorithm to restart 10 times using a different random starting location for each of the 3 centroids each time. We could set the number of restarts higher to potentially get better centroids for our clusters at the expense of the amount of time it takes to run the algorithm, however, after a given number of restarts, you generally won't see much additional improvement by adding more random restarts. What the K-means cluster algorithm is doing is attempting to find the appropriate centroids for each of the 3 clusters in the 4-dimensional space of the data set. How the algorithm does this would take a bit of time to explain, but since this is just a teaser to show you what else is possible in R, we're going to skip over the details and go right to the end results. We can visualize the output of the K-means cluster algorithm by redrawing our scatter plot using separate colors for each of the species, but instead we'll also add a shape for the predicted species produced by the cluster algorithm. We'll do this by setting the plot character, that is the pch argument, equal to the integer id of the predicted cluster for each point on the plot. So now in the case of our scatter plot, black still represents actual setosa flower, red still represents actual versicolor flowers, and green still represents actual virginica flowers. However, circles represent predicted setosa flowers, crosses represent predicted versicolor flowers, and triangles represent predicted virginica flowers. As we can see, all the setosa flowers were correctly classified, which are represented by black circles. Most of the versicolor flowers are correctly classified, that is the red crosses, and most of the virginica flowers are correctly classified as well, that is the green triangles. However, some of the versicolor flowers are incorrectly classified as virginica, that is the red triangles, and a few of the virginica flowers are incorrectly classified as versicolor, that is the green crosses. All in all, the clustering algorithm did a fairly good job of classifying the flowers by species based on the available data. If we're interested in seeing the centroid of each cluster plotted in this 2-dimensional projection of our data set, we can plot the centroids as points on the scatter plot. We'll do this with the points command and we'll set the x coordinate equal to the x location for each of the cluster centers, set the y coordinate equal to the y location for each of the cluster centers, set the plot character to 4, which is an x character, set the plot with argument to 4 to make the plot character more visible, and set the color to blue to help us easily distinguish it from the other plots. As we can see, this draws thick blue x's at the center of each cluster in the scatter plot. Finally if we're interested in seeing how many flower are correctly classified versus incorrectly classified, we can create a contingency table with the species on the columns and predicted clusters on the rows. As we can see, all setosa flowers were correctly classified into cluster 1, 48 versicolor flowers were correctly classified into cluster #3, however, 2 versicolor flowers were incorrectly classified into cluster #2. Thirty six virginica flowers were correctly classified into cluster #2, however 14 virginica flowers were incorrectly classified into cluster 3. While the K-means cluster algorithm wasn't able to perfectly predict the correct species for each flower, other more powerful machine learning algorithms, like linear classifiers, neural networks or support vector machines might be able to do a better job. While I would love to go much deeper into cluster analysis as well as data mining and machine learning, unfortunately we would need an entire course to go much further. So we'll conclude our cluster analysis demo as just a glimpse into the statistical pattern recognition that can be performed with R. If this type of analysis interests you though, I highly encourage you to learn more about data mining and machine learning. So now that we've helped Joe get his new business up and running using our data analysis skills, he leaves his position as CFO and moves on to pursue his dream of running his very own, high end flower shop catering to the rich and famous in Hollywood. He takes us out to celebrate his grand opening, his flower shop is a huge success and Joe lives happily ever after.

Alternatives to R

So now that you've seen how to use R for exploratory data analysis, you probably either love it, you hate it, or you might just think it's okay. In the event you're in the hate it camp, but you still really like this idea of doing exploratory data analysis, no worries because there are plenty of other ways you can do exploratory data analysis without R. Let's take a look at a few of these alternatives in order of least difficult to learn to most difficult to learn. First, spreadsheets like Microsoft Excel are still probably the most popular choice in the business world for doing exploratory data analysis. Spreadsheets are quick, interactive, and they have a minimal learning curve. If you'd like to give Microsoft Excel a try, you can download a free trial at www. office. com. Next we have interactive data visualization tools, like my open source software project called Data Explorer. It's highly visual, highly interactive, and it's very easy to use even for users with little to no experience with data visualization. If this looks interesting to you, you can download a free copy of Data Explorer at www. data-explorer. com. Another step up in complexity, we have more advanced interactive data visualization tools, like Tableau. This is like my open source Data Explorer, but with much more advanced data visualization and analysis features. However, Tableau desktop is commercial software, that is it's not free, open source software, and it takes a bit of time to get up and running. If you'd like to give Tableau a try, just visit www. tableau. com. Moving further up in complexity we have statistical analysis software with rich user interfaces like SPSS. These provide much of the power of R, but in a much more user friendly interface. Choosing a program like SPSS over R has a lot to do with personal preference and whether you enjoy working with programming languages or not. You can download a trial copy of SPSS at www. ibm. com/software/analytics/spss. In addition to R, we also have other statistical programming languages like SAS. These are similar in power and complexity to R, however, most are not as popular as R and many are not free like R. Languages like SAS, however, are still heavily used in the business world for data analysis. If you'd like to learn more about SAS please visit www. sas. com. And finally, you can use other general purpose programming languages, like C#, Java, or Python, to do all the same things that you can do in R. R, however, was designed for data analysis, so you may gain flexibility with general purpose languages, but at the expense of powerful data manipulation and analysis constructs. What I recommend is to learn R in addition to your primary general purpose programming language like C# or Java. You'll still use your primary language for building for most software, but you should use R when you're specifically working with data. Having R as an additional tool in your programmer tool belt is a very powerful way to increase your value as a software developer.

Summary

In this module we learned about other types of data analysis that we can perform beyond exploratory data analysis. Next we saw a demo of linear regression analysis and cluster analysis, then we learned about a few alternatives to R for performing exploratory data analysis. Now let's take a few minutes to wrap up things for the course as a whole. If you'd like to learn more about the topics we discussed in this course, I have links to a few resources you might be interested in. First, I recommend that you head to the R website and download a free copy of R, this way you have it installed and ready to go when you're ready to start working with data. Next, head to the R Studio website and download a free copy of R Studio. With R Studio you'll get up and running much quicker than by using R out of the box. If you're interested in additional online training, Pluralsight has other courses on both R and data analysis available and coursera has a data science specialization track taught by professors as Johns Hopkins that provides in depth coverage of using R for many different types of data analysis. For blogs and online articles, Nathan Yau's flowing data website is an excellent source for learning about data visualization with R. Revolution analytics has a blog called revolutions, which provides the latest industry news about R and R-blogger is a great site for keeping up with the R community. Finally, one last thing before we wrap things up for this course. Feedback is very important to me. I use your feedback to improve each and every one of my courses. So please be sure to take a moment to rate this course, ask questions in the discussion board if there's something in this course that you didn't understand or would like me to clarify, and leave comments to let me know what you liked about this course or if you think there's something I could improve upon for future courses. I would love to hear from you. To summarize what we've accomplished in this course, first we learned about the R programming language, why it has become so popular for data analysis, and how to perform basic programming tasks in R. Next, we learned how to use R to load, transform, clean, and export our data. Then we learned how to calculate descriptive statistics for our data to quickly summarize our data in numerical ways. Next, we learned how to visualize our data using the basic R plotting system. Finally, we looked at several topics that go beyond R and exploratory data analysis. This course would not be possible without the help of many people that were involved in the creation of this course. I would, however, like to give special thanks to Nathan Yau of Flowing Data for allowing me to use several of his advanced data visualizations for this course, Jeff Leek and Roger Peng of Johns Hopkins University for teaching me quite a bit of what I know today about R and data analysis. Karen Bittner, Anne Herlache, and Cory House for providing me with technical feedback on all of my course materials and I would especially like to thank you for joining us for this course and for sticking with it until the end. I hope you've learned some valuable new skills that you'll be able to put to great use and I hope to see you again in another Pluralsight course in the future.

Course author

Matthew Renze

Matthew is a data science consultant, author, and international public speaker. He has over 17 years of professional experience working with tech startups to Fortune 500 companies. He is a...

Course info

Level	Beginner
Rating	★★★★☆ (327)
My rating	★★★★★
Duration	2h 30m
Updated	13 Nov 2015

Share course