**Importing CSV Files: Video Transcript**

There are various ways in which we can import our data into R. And which one we use depends on what type of file we're reading in and also what R package you would like to use. In this video, we're going to look at how to import one of the most basic and common types of file that you come across, a CSV file.

Csv stands for comma separated value and it is a very uncomplicated type of file, a plain text file. Plain text just refers to the fact that it doesn't store any information about whether the text is bold or italic, or any information about colors or tables. It's just a record of the raw data in it's simplest form. It's simplicity makes it easy to share these files between different types of applications. You might have come across CSV files when using Excel. When you open up a CSV file in Excel, it looks just like any other Excel file. But as I mentioned, the difference is that there'll be no formatting. If you do apply formatting and excel, say for example, highlight some of the text or create a graph, and you want to then save these changes, you will need to save it as an Excel file instead of csv.

So how do we load a CSV file into R without loading any additional packages? We could use base R and called the command read.csv(). Base R refers to the default set of packages which were automatically installed when we first installed R. There is however, a more efficient way of loading CSV data than using the read.csv() command. And this uses a very similar command, read\_csb(). This command comes from the tidyverse set of packages, which you've already come across and more specifically, the readr package. So let's start by loading the readr package. You'll hopefully already have installed this package when you installed the tidyverse in a previous lesson. But if you haven't, go back and make sure you do that first. Now we're going to load the package by typing the command library(tidyverse) and then hitting Ctrl and enter. We could have got away with loading just the readr package but since we will be using the tidyverse set of packages extensively through this course, it makes sense to get into the habit of loading the entire set at one go at the start of your scripts.

Now I've already pre saved a CSV file of cancelled operations data in my Project folder. So I'm going to read that in. We would advise that you always work within a project as this means that R and find your files more easily. You can find further information about how to set this up in previous lessons. A quick way of checking that you're in a project is to look up here. We can see that we are in the project cancelled operations. So we're good to go. We can read in our CSV file by using the command read\_csv(), and then the name of our file. The name of our file must be in within quotation marks. And then control and enter - success.

Hang on! It's appeared down in the console and we want it over in our environments tab so that we can use it in our code. That's because we've not saved as an object. Okay, let's go back up and give it a name. Cancelled ops One. Now when we run it, it appears over here in our environments tab under the category data. Excellent. We included only one argument in our read CSV command or if you like, one bit of information within the brackets. This argument is the path or location of the file. We can get away with just the filename here because R already knows to the look inside the projects file. This is one of the great advantages of working within an R project. But if we are working on a big project with a number of data files, we might want to put our CSV files in a separate data folder inside our project folder, for example down here. It's not so obvious now to R how to find this. So we would have to add in this bit of information about the file path to the read\_csv command. Let's copy and paste our code above Control-C, Control-V. So we're going to read in the second dataset to change that to two. But we have to tell R that is in the data folder now. So I put data and then forward slash. And now when we press control and enter, it works.

It's appeared over here, but different operating systems deal with paths in slightly different ways. So a script that works on your computer might not work on someone else's. And so an easier and more robust way to let R know where your files are is to use the package 'here'. Yes - the packages actually called 'here'! So let's add this package to the top of our script beside the tidyverse library here. And then control and enter to run that line. Oh no! Down in the console here we're seeing an error message. It says that there's no package called here, but we know there is, so it must mean that we've not installed it yet. That's okay, we can install it. Now you might remember that the command we use to install packages is install.packages and will only need to install this package once. We won't need to save this bit of code. So let's use the console install. A little tip here. As I type install, you can see that we're being given suggestions as to what we might be looking for. And if the correct suggestion is there, we can just hit Tab to complete the command and save us some typing. And then here, and again, as I type here, we can see a suggestion. And I've just realized I forgotten the quotation marks. That's okay. If I press Tab, my code has been corrected for me. Brilliant. And then we hit Return to run this code. We don't need to press control and enter here, just enter because we're in the console. And now if we go back up here to run the library command for here, we don't get an error message and it's been loaded successfully. Great.

Now that we've loaded the here package, instead of writing the path name with the forward slashes, we can instead use the following code. First let's copy Control C. Control V. We're going to read in the third data set. We need to add in here command and the first argument is going to be the name of the folder. So that's data, and the second argument is the name of your file. As before, let's check. We've got the right number of brackets, 1,2.... We're missing one at the end here. This code will work on any computer regardless of operating system. And we don't have to worry about where to put the slash in either. Let's run the code and see what we get. Control and enter. Great. And if we click on it, we can see it's been loaded. The output looks like a table, but it's actually a special type of table that the tidyverse cause a TIBBLE. Another cool feature of RStudio is that it has a data import wizard, which could help us if we forgotten the code to read in our data or if our file is a bit messy, it gives us a chance to preview it to see how R is going to cope with importing it.

So let's try this alternative. We've got a fourth dataset, so let's try and install this. If we click on the name, this gives us two options. We can view our file, which just lets us check the contents. But the data has not been loaded into R yet, so we can't do anything useful with it. We need to choose import option, which lets us see a preview. Down here we can see some familiar looking code. Our read Csv is there, and the correct file path too. We've also got information on what datatype R thinks each column is. Our months column we would ideally like to turn it into a date data type, but R thinks it's a number. The word double here is one of R's number data types. So let's see if we can change this to a date. If we click on the drop-down and select Date, it wants to know what format our date is currently in, and so we can tell it it's a four-digit year. That's what the percent and capital Y stands for, followed by a two-digit month. That's the percent sign and the lowercase m. And there are no separating characters, so no slashes or anything. Let's see what happens when we press OK. Amazing that worked, and now it's in a better format.

R has added the first of the month each date, as it didn't like having only a partial date, but that's fine as long as we're aware of that. The next thing we notice is that there are columns without any data. So if we are sure they're completely empty, we can just ask R to skip these by going to the same drop-down menu and clicking skip. There are various other options down here which we can check, but for this dataset, the default values look good. Now we have the option to import, but we don't actually want to click on it because all of the changes that we made to format the data have appeared as if by magic in our R code preview. And although the data will be imported, the Settings which we applied to import it will be lost if we click on the Import button. And so if you want to load an updated data set in a month's time, say we would have to go through this whole process again. Instead, the smart option is to copy the code into our R script so that it can be used again and again. And we can do this easily by clicking on the clipboard icon. And then if we hit cancel and back in our script, we can copy the code and Control V. Now two lines of code are redundant and we can delete these. So library readr we don't need, as this is included in the tidyverse set of packages. So let's delete that and the view command down here, it's not necessary because we can always click on the data from the Environment tab if you want to have a quick look. So let's delete that too. Now let's check that it runs and there it is under data. And if we click on it, we've got a nicely formatted table, with the date column now with the correct data type and the columns without data have been removed too. What a great feature.

Now try load again some of your own messy CSV files to see how they look and see how well R does at guessing what data types you have.