Applied Predictive Modeling

Lab 1, Part III Transcript: Introduction to R – Useful Functions

Let's go to reading and writing files in R. So one way to look at what directory you are in, in R is this nice get working directory function. So when you do when you run this, it tells you, hey, right now I'm in this folder, Math 503 Module I. And then you can set working directory, whichever directory you want.

So you can do set working directory and then give the part of the file. So this may vary between Windows and Mac. So just be careful so you may need the double slash and other things on Windows. So this is what works on the Macs and Mac and Linux kind of OS environments. And then once you've set the working directory, so there is this neat function called read_csv. So I have this function hcvdat0 So this is a dataframe that is available to you as part of Module I homework so you can reference this file there. So I'm just reading that file here.

So read df1. I'm assigning it to this variable, read_dataframe1 and then read.csv reads the CSV file. Now, we can do the same thing that we did last time. We can look at the structure of this file. So one thing you can see that's interesting here is this is a dataframe of 615 observations and 14 columns. But like the thing that we observed previously when we created the dataframe, the strings has factors is equal to false is something that comes up here also.

You can clearly see that there is these columns category and sex that are factor at this point. You may not want them to be a factor and you may want them to be characters. The way you can do it is similarly like we did for the dataframe. We just say strings as factors is equal to false and then that's going to create a new dataframe. And if you look at the structure of this, now you can clearly see it has changed to character. With our versions 4.0 and above I think strings has factors is equal to false by default but it was true by default in the prior versions of R .

Depending on the version of R you are using, you just have to be careful what this argument is set to. So analogous to the read.csv, you have write.csv function that helps you write CSV files and dataframes into a file. So here, I have the write.csv, name of the dataframe that we created somewhere here. And then I'm saying hey, write it to a file named xds2.csv. So if I execute this function, it's going to just go and write this function in the directory that you are in R you can give it a path and then ask it to write it in that location. To know more about write.csv, you can do... you can look at the help. For the write.csv, it has a lot of nice arguments that you could use to write the file in a specific way that you want.

I won't go too deep into write.csv. It's pretty straightforward. You can research the help page and get more out of it.

So next, I think we'll go into the apply group of functions. I would say this is one of the most handy set of functions that you'll learn in R because they help you avoid using loops and you can also write very clean code using it. So to learn more about apply functions, I'll put a link in here that you can go and learn more about it but I'll give you a basic flavor of the apply functions here.

So, let's use a dataset called cars, data cars.

There is a specific set of datasets that come by default with R installation. So one of them is this cars dataframe. So, and you can load it by doing data cars. It creates a dataframe cars. And then if you do the head cars, you can see that this is a dataframe with this because you did the head, you get six rows. it has two columns: speed and distance. And then if you look at the dimension of the cars, you can see it's a 50 observations and two columns.

The first function that we look at is the apply function. So this apply function takes an object like a matrix or a dataframe which is two dimensional in nature. You have a row and column in matrix. Also you have rows and columns.

And then the next argument for this apply function is the axis along which the apply has to apply the function. So in this case, one means it's across rows. Two means across columns and the function to apply. So here I'm saying apply mean function to these cars, dataframe, across rows. So it'll give like a row mean.

Because we know there are 50 observations, when I execute this, I should get 50 values. These are row average, row averages because those are row means. Similarly, if I do the same thing, cars to mean, I should get column averages because we only have two columns. It should only give you two values which is the speed and distance and those averages. You can see speed and distance. You have an average of 15.4 and 42.98.

So this is very nice where you don't have to worry about it. However many columns are there, however many rows are there, this function nicely gets you what you need. And then similarly, we can use another dataset called air quality. Again, this data is called air quality. And if you look at the head, you can look at it has six columns and you can also see there are some NAs here. And then because I did the head, you only see six rows. And then the next function is the lapply function. Lapply function is kind of similar to apply function but the L stands for the list. So it outputs a list when you apply a function to an object.

The arguments are the object and the function you want to apply to that object. So this object could be a dataframe or a list of objects where you want to apply this function to every element of the list. So in this class I'm saying, hey, we have this air quality dataframe. I want to know the class of each column of air quality.

If I execute this and then look at AQ class, so you can clearly see I got the class of each of those. And you can also see the dollar which indicates that your object AQ class is a list. You can also see it here. So it outputs a list and you can see that we have integers and the numeric vectors in here. And then AQ means, so here I can do the same thing. Instead of class, I'll apply the mean function to see what is the mean of this mean of each of the columns in the air quality.

And then if I execute this, you'll see it was unable to compute the mean for ozone and solar but it was able to compute it for all other columns. The reason this happens is because you have NAs in the dataset. So when you have NAs in the dataset, the mean function does not work as you expect it to work. But there is an argument to fix this problem. So how do we take care of this NA issue when we are calculating mean? So the mean function has an argument called NA.RM. So if you say NA.RM is equal to true, then it's going to ignore those NA values and compute the mean. That way, if your data has NAs, you don't have to worry that you won't be able to compute the mean.

Let's see what happens if we do this. So if you check here now, you can clearly see the ozone and solar which did not have a mean before have a mean now. So that problem has been resolved.

And then you can also use lapply sometimes to get a subset. So, one example I want to give you is like let's say you have air quality and you only want columns that are integers in your data. We know that if we look at the air quality, everything except wind is an integer.

If we want to subset something like that, all we have to do is we can do air quality and we know the first value within the dataframe is the row selection and the second one is a column selection. So here I'm saying I'll apply air quality class is equal to integer so you should get a new dataframe, air quality_s. So you can see that instead of six, we have five variables. It should have removed the wind variable which was not an integer. So you only see integers in this case.

It's a nice way of selecting something in a single line of code rather than iterating through and then adding every column by checking if each column is a particular thing or not. And similar to lapply, there is a function called sapply. Sapply is just a wrapper on top of lapply. So it works very similar to lapply, but it can... Its output is a more simplified version of it. It usually gives you a vector or something else. So, other than that, there are no big difference between sapply and lapply. Lapply outputs a list always whereas sapply kind of simplifies the output to what is most reasonable. So usually, you may get a vector or just values as a vector. So as an example, I can show you here a sapply. So I'm, if I want to find how many NAs are there in this because we know the air quality had some NAs, so we can pass air quality. And then this is like a Lambda function in Python. So you can create a function, function X.

It takes is .NA is true, If there is an NA value and is false. If there are no NAs, we are finding the sum of all those NAs. So what happens is if I run this? For each column it's going to compute all the number of rows that are NAs and we can see ozone and solar are the ones that have NAs and remaining columns don't have NAs.