R Programming Fundamentals

by Abhishek Kumar

Getting Started

Course Introduction

Hi, this is Abhishek Kumar and welcome to this fundamental course on R programming. This is the first module in a course that will show you everything you need to know to start working with R framework. In this modern world, there is a surplus of data originating from various sources. Whether it's raw data, or financial transactions, or geographical especial data, there are plenty of data sources. But, we need some technique to extract meaningful information from this heap of data. If you are not able to extract meaningful information or knowledge, then it doesn't make any sense. This leads to the importance of data analysis and data science in the present world. I would also like to quote a famous line, used by industry experts across the world, that data scientists can be termed as the sexiest job of 21st century due to it's importance in the present scenario where companies want to extract business insights using data to take various key decisions, which in turn, can improve their bottom line. So, if you are also interested in data analysis field snd want to build a carrier in this. Then, this course will surely help you to take the first solid step. This course will create a base foundation for R framework. R is a free and open source language environment. It can help you to perform various data analysis tasks quite efficiently and easily. All the period R language has become defacto standard by staticians and data analyses across the globe. Our language is gaining popularity day by day and a complete ecosystem is getting

developed around it. This fundamental course will guide you through the R language and its capabilities. You will learn various core programming concepts in this course to become an efficient and productive R programmer. Other than code programming features, you will also learn to do some basic data analysis with the help of R language where you will learn to play with data. As I mentioned earlier this course is aimed to provide a solid foundation to be en effective R programmer. This course doesn't require any specific prerequisite so if you are aware of any computer programming you are good to go. In this course, I will not be covering each and every aspect or feature of our framework or our programming. Rather, I'll focus on the topics which are absolutely essential to get you going in the R Programming world. Moreover, I'll try to answer the questions like what a certain construct is and why do you want to use it. I'll be using real world examples to further solidify the concepts. The course contains lots and lots of demos so that you can follow along with me throughout this course and learn the concepts easily. So, this course will surely help you to take the first solid step towards learning R programing, and thus the worlds learning data science.

R Overview

Before getting into the details, let's have an overview of R framework as a whole. R was developed on the base of S programming language, which was originally developed by John Chambers, when he was in Bell Labs. Then, R was mainly developed by Ross Ihaka and Robert Gentleman at the University of Auckland. They developed this as a free statistical involvement to teach their students, but when it started gaining popularity they promoted this R language. Currently, there is R development core team which is taking care of its growth. In fact, John Chambers is also a member of this core team. R is generally treated as a statistical software package with great statistical operational capabilities. Moreover, this free and open source Statistical package also compares well with other popular commercial packages; such as SAS, and SPSS'. Over the period R has evolved as a free programming language and environment. And now, R is not only used for statistical purpose but it is used in variety of applications. And, I list out a few of those applications in upcoming slides. Let's see the timeline for code R development. R developed started in early 1990s. Another version was released in 1997. After that version 1. 0 was released in the year 2000 and version 2. 0 in year 2004. And, in 2013 version 3. 0 was released. So, in this course we'll be working with version 3 of R. In each, each major and minor revisions, two features have been added and the bugs have been fixed. So, if you want to know more about the changes in each version, you can visit this link.

Why R?

So, what makes R so special and a favorite framework among data analysts and data miners across the globe? Probably the first reason is that you do not have to spend a penny to work with R framework. It is completely free to download. It is an open source language under GNU General Public License. And, because it is open source, not only you can access those source code, but also you can tweak it as per your requirement. And, due to this very nature, R has been improved over the period by double uppers across the row, and has become very stable and reliable. Moreover, R is cross-platform compatible. It is at a level for all major operating systems. So, if you are a Windows user, Mac user, or a Linux user. R will be available to you without any extra effort. Another very important aspect of R is its extensibility. It is very easy to double up packages in R framework. You can easily create and publish a package. Due to this feature there are thousands of package already available for different kinds of tasks. In fact, as on date, there are over 5, 000 packages available which are developed for different kind of applications. These packages are also free, so you can use them in your applications also. For any framework to become successful, it is very important to have an active community. Where users can share their problems, get solutions, and share their experiences and in the case of R there is a very active community. There are mailing lists, forums, blogs, and all kind of supports available to everyone. I'll show few of these resources in the next module. Another major benefit of using R Framework is its ability to integrate with other languages and packages. So, you can integrate your code written in C, C++, Java, Python, and other popular programming languages with R Framework. It can also be integrated with other commercial packages, such as SAS and SPSS. You can also connect R framework to other database systems such as SQL server, and MySQL. In fact, R Discus Vegas database connectivity options in the later part of this course. So, as you can see that, over the period a complete ecosystem is getting developed around R framework. And, due to its versatility and usefulness, R has become a preferred language and framework for statisticians, data analysts, and data miners across the globe. There is an increasing trend in R popularity in the recent times. Let's look at a few reports to support this trend. According to Rexter Analytics data miner survey in 2013, R has emerged as one of the most popular framework among data miners across the globe. This graph depicts users familiarity and usage of different data analytics frameworks. As you can see here, R fairs really well with respect to other frameworks. According to the survey around 70% data miners reported the usage of R in their activities and approximately 24% of data miners use R as a primary tool for data analysis and their work. You could also see the increasing trend over the past few years. These trends reflect the increasing popularity of R among its users. Not only this, I is becoming a preferred framework in radius data science competitions also, such

as Kagle. Kaggle is a very famous predictive analysis competition platform where radius companies post real world problems to data scientists around the world. And, data scientists try to solve these problems, and participants can win big prizes also. And, R is a very popular framework among data scientists taking part in Kaggle competitions. In fact, in a presentation on Apple 2011, Kaggle's CEO, Anthony Goldblum, mentioned that R is used by around 30% computers across the globe. Moreover, Kaggle also reports that 50% of competition winners used our framework to beat out their competitors to create the most accurate predictive models. So, as you can see that R is a very popular, as well as very powerful framework. Nowadays, R is used to perform various kind of tasks. As I already mentioned, R was developed with the intention of using it for various kind of statistical task. But other than a statistics, R is used in other areas also. Art has great support for Visualization. You can create various kind of interesting plots and images using our frame book. There are various packages already available which can perform the task of data visualization quite efficiently and easily. R is also used in various kind of machine learning tasks such as classification, clustering and aggression. There are various optimization packages also available in R, which can be used for various kind of optimization problems. Nowadays, R is also used to process geographical or spatial data. Art has also been used in high performance computing, or panel computing. In fact, latest packages are deadlocked to process large data sets in panel environment. Although, I have listed out only a few task here, but this will give you an idea how effective and useful Art framework can be. Due to this ability of R framework to perform various kind of tasks. R is used in variety of applications such as various kind of financial modeling problems, market research problems, social science studies, natural language processing, genetic studies. And, the buzzword of today's time that is big data. I have mentioned only few application areas here but possibilities are endless. So, if you're working on any kind of data analysis problem, our framework can be a good alternative for you.

Why Learn R Programming?

There are various graphical user interface based applications out there for various kind of data analysis tasks. In fact, many GUI based applications are also built on R framework. These applications can perform various data analysis tasks with few clicks. Then, why would you even both to learn R Programming? What benefits will you get by adding this programming skill to your repertoire? First of all, if you have to perform specific tasks, which are not provisioned in the available GUI desktop application, then you do not have any other option than to create your own program to do your work. And, not only that, with the help of programming you can create repeatable, or reproducible steps. So, every time you run the code you can be assured of the task

completion. This could be very handy for bigger data analysis projects. For GUI based applications, this could be a very tedious task. Moreover, as R is an open-source framework, so source code is also available to you. You can also look at the source code of various packages developed in R. So, if you know R programming, then you will be able to understand the intricacies involved in their implementations. And, if the existing implementation doesn't suit you needs then you can even extend or modify the existing implementation. Your R programming skills will also be used if you want to write software for this kind of data analysis tasks. So overall, if you aspire to be a productive and effective data scientist. You should have a good command over R programming language, believe me or not. It is going to help you in the long run. So, till now, you must be having a good feel of R framework and its capabilities. Now, let's see the simple procedure to install R on your own machine. So, that you can work along with me in the demos to have a good hands on practice. In the next clip, I'll walk you through the simple steps of installing R on your Windows machine. I have also included clips for R installation in Linux as well as in the Mac operating systems. So, depending upon your enrollment, you can go to the required clip.

Install R on Windows

To install R on your Windows machine you can go to this webpage. This is the webpage for Comprehensive R Archive Network, which is popularly known as CRAN in the R world. To come to this webpage, you can use your favorite search engine. Here, I'm using Google. Just enter cran in your search box and most probably you will get this link as the first result. So, on this website you will have most of the information you need to get going with R. Here in the top, you can see the links for downloading R for Linux, Windows, and Mac. However, as here I'm on a Windows machine, I'll show you the installation of the Windows version. But installing R in other operating systems is also very easy. If you are a Linux or Mac user, you can view the respective installation clip available in this module. So, for Windows user, click this link, download R for Windows. This will take you to this web page. Here you will see three subdirectories. Here, as we are installing R for the first time on our machine, we will go to the base subdirectory. This will install base R framework and some essential packages. So click on this base link. Here on this webpage, you will see the download link for R. As you can see at the time of this recording, we have the version 3. 1. 0. Here, you will also find installation instruction link, and what's new in this version link. So, if you want to know more about this version, then you can visit this link also. Now, let's download R by clicking the Download link. A file down request will be popped out. Let's download and save it on our desk. Here I have copied the downloaded file to my local folder. So to install R, you simply have to double click on this exe file. Select the language. Click Next. Next, Next, Next, Next, Next,

Next. Here I'm not alternating any default settings, so at the end of it R will be installed on your machine. In Windows machine R will be installed with R GUI also. To access this, you can search for R in your installed applications. Here I am on a Windows 8 machine. I'll search R using search feature of Windows 8 charm. If you have a 64-bit machine, then both 32 and 64-bit version of R will be installed. So, to access 32 with R, type R space I. Here you can see we have R I386 as a result, click on this. This will launch the 32 with RGUI. inIthis GUI you can see, we have a console window. You can see the version details of your installed R framework. Here it is mentioned that this is a 32 bit version. Similarly, to open the 64 bit version type R space X, and here you can see you have a RX64 result. Click on this. This will launch the 64 bit RGUI. To confirm this, you can see the version detail. In the next two clips, I'll demonstrate the installation of R in Linux and Mac operating systems. So, if you are a Windows user, you can skip the next two clips.

Install R on Linux

Installing R in a Linux environment is also very easy. To get all instructions for installing R on your Linux machine, you visit this CRAN webpage. To come to this webpage, you can use your favorite search engine. Here I'm using Google. Just enter CRAN in your search box and most probably you will get this link as the first result. Coming back to this CRAN webpage, you can click Download R on Linux link. Then you can drill down to your respective Linux flavor. I have Ubuntu installed on my machine, so I will click on Ubuntu link here. Here you will find links for different versions of Ubuntu. However, in this clip, we will install R using Command Line. For this, instructions are also written here. You can follow them to easily install R on you Ubuntu machine. There are three main steps involved in this process. In the first step, we will add the repository from which we can get the latest R package. So the repository link you want to use depends upon your choice of CRAN mirror. Mirrors means identical R packages are available on different geographical locations. You can select the mirror nearer to your place. To see all mirrors, you can click this link. Here you can see different CRAN mirrors. We will be using this first mirrored link. Let's go back. The repository link for Mac is as follows. Depending upon your Ubuntu version, you can choose the format of your repository link. Because I have Ubuntu 12. 0. 4, that is Precise Pangolin, so I'll use this format here. After adding this repository, you can use these two commands here to install Base R on your machine. These two statements are your second and third step. Let's see these steps in action. First open the terminal. Here's the command to add the repository. If prompted for password, then you can enter your user's password. So this was our first step. In the second step, we will use the update command of app get. Here's the command. Hits Enter. This will download the packages list from the repository and update them to get the information on the latest

versions of packages and their dependencies. If you scroll a little bit, here you can find that this command updated the package list from the repository which we had added in the previous step. So now we're done with the second step, we can move ahead with our installation in third and last step. We can use this hud statement. This will install the base R version on your machine. Press Enter to continue. If prompted for confirmation, enter y to continue, y again. It will download the latest R package and will install everything. It will take a while, so let's skip to the end of this step. So now, our installation is completed. There is no RGUI in Linux, which is available in Windows and Mac. But it can access out through your terminal itself. For that, you can simply type capital R in your terminal, press Enter. So as you can see here, R is up and running on your machine. You can see your R version here. We have version 3. 1. 0. So now you are all set to go ahead. In the next clip, I'll show the installation of R in Mac operating system. So if you are a Linux user, then you can skip the next clip and automatically jump to hello world with R clip.

Install R on Mac

Installing R in Mac operating system is also very easy. To get the required installer of R for Mac, you can visit this CRAN link. To come to this web page, you can use your favorite search engine. Here I'm using Google. Just enter CRAN, CRAN in your search box and most probably you will get this link as your first result. So on this website, you will get most of the information you need to know to get going with R. Here on the top, you can see the links for downloading R for Linux, Windows and Mac. As here I am on a Mac machine, so I'll show the installation of the Mac version. But installing R on other operating systems are also very easy. If you are a Linux or Windows user, then you can watch the respective installation clips also available in this module. So for the Mac users, click this link, download R for Mac. This will take you to this webpage. Here you will find some PKG files. Now, depending on your Mac OS version, you can select the respective installer file. Because I have Mac OSX 10. 8 Mountain Lion on my machine, so I'll be downloading this first link here. So just click this hyperlink to download this file. Depending upon your internet connectivity, it can take a while to download this file. I have already downloaded this file. Now I can simply double click this file to start the installation process. Click Continue. I'll be taking all the default options here. So just click Continue again. Continue, click Agree to accept the terms. Now click Install. It will prompt for a user password. Let's enter that Now, this installation process will take a while. And after some time, R will be installed on your machine. So now the installation was successful. Close it. Now, to launch the R application, you can go to applications, then search for R. Here it is. Now, click this icon to launch R. So here on the console window, you can check your

R install version. So now R is installed on your machine, and you are ready to go. In the next clip, I'll show you some hello world stuffs in R.

Hello World in R

Once you have installed R on your machine, you are good to go further. This clip will show you some hello world stuffs just to get you going. Currently, I'm on a Windows operating system and I have RGui with R Console at my disposal. However, if you are working in Mac or Linux environment, then also you can follow along as I'll be mainly using the R console in this demonstration. As you can see here, that R Console is starred with some important information, such as it's version, then 32 bit or 64 bit information. There are some license related information also. Then we have some commands to get contributor's list. Finally, a few more commands to get some help and how to quit R. Then we have a prompt here where we can enter R commands. Now, let's enter our first command. Let's print hello world itself. We can use the print function for that. In this case, we can parse the string as an argument. So we have parsed hello world wrapped in quotes as the argument here. Now press Enter to execute this command. As a result, we get the hello world equaled back to us. So this was our hello world example. However, as R is a powerful tool, so let's do some more work. Here we have used a mean function. Inside this mean function, we have parsed sequence of numbers. To generate sequence of numbers between one and 20, we have used a sequence operator, which is the colon sign here. This will generate numbers from one to 20 with the default step size of one. Let's exude this statement. Press the Enter key for this. So we get the mean value for the numbers from one to 20. As you can see here, with a single small line of code, we are able to get the mean value. If you have worked in any other traditional language, then you know that it will take more work to achieve the same result. Probably you can use some looping mechanism to through the numbers, then adds those numbers, and finally find the average and mean value. But here, we accomplished the same job in the single line. So in this example, we made use of colon R sequence operator. Another important operator I want to discuss right away is the assignment operator. Suppose you want to store the sequence of numbers from one to 20 into some variable, then you can do something like this. Here you can think x as some kind of container to store some stuffs. And we are putting the sequence of numbers from 1 to 20 in this container. And for this operation, we have used this assignment operator, which is this less than sign followed by the dash sign. Let's enter to execute it. So once you have put some values in your variables, you can simply use that variable. Let's do it now. Here we are evaluating the mean value of this x variable, which contains the sequence of numbers from 1 to 20. So you see, here we get the same result back. Well, this is just the

beginning. And throughout this whole course, we will look at various other constructs of R framework and R language. One thing you might have noticed that so far we have written four commands. Each R command one by one in this R console. This could be good for very small tasks, but in the real world, you would like to have a script where you can write different R commands in some particular order. And you would like to execute either some portion or all of that script. Well, if you are working in Windows environment, then you can use RGui to create such scripts. You can click File and then New Script. You will get a script window. And here you can write your script. In a similar fashion, Mac users can click File and then New Document. And here they can write the R script. For Linux users, there is no such GUI application. And because I want all Windows, Mac, and Linux users to follow along with me, I will rather use a simple editor, such as Notepad on Windows to write the script. If you are a Linux or Mac user, then you can use any text editor you are comfortable with. So here we are using Notepad application on a Windows machine. Now let's put all four line of codes here. Now let's save the script. Let's give it a name, myscript. We have deliberately used R as an extension to save this file. But even if you do not use this R extension then still it will work, but it is a good practice to use R as an extension for R script files. So now our script is done. Next step is to import that script into our R console. To do this, we will use another R command, which is source command. So we will write in the console. And because we use source command to import R script in the console, we call this process as sourcing the script. Sourcing a script is just like sending the whole R script to the R console. The first parameter of the source command is the file path. Here I have used absolute part of the script. We will talk about absolute and related part later in this course. This source command has several other parameters also, but we really care of them as of now. Other than this equal parameter. I have deliberately set it to true so that we can see the result of each line of the script. So to execute this script, now we will hit Enter. So as you can see, all four lines of code are executed one by one on its own. So in this clip, we discussed some hello world stuff. Moreover, in this demo we used a very primitive editor. But editors are very important if you're working on a large project. Therefore, we will talk about editors, IDs, and their requirements, in the next clip.

Editors and IDEs for R

So as you saw in the previous demo, you can write commands in the console and get the results back. But if you really want to get the benefit of R programming, you will be writing codes. And for writing codes, you need some good editor. We have already seen one such editor in R-Gui. Which comes with a base installation of R in the Windows environment. Similarly for a Mac machine there is a default editor in the Rapp. For Linux there is no dedicated editor for writing

codes. And you can use your existing editors like RIM, or Emac to write R programs. However, there are other editors and extensions also which can help to write your code easily. I have listed a few editors here. If you're already using Emac editor, then you can use this add-on package I've labeled on this link. This will install the ESS add on for your Emac editor. ESS means Emac Speaks Statistics. You can use this add-on if you are working with other statistical language also, other than R, such as S+ or SAS. If you are concerned with Vim editor then you can use the extension downloadable at this link. There are other options also available to you. If you are a fan of Notepad++ then there is extension NPPToR available to use. You can also use TINN-R editor if you want to. So I have listed out many editors for you and these are pretty good if you are working on smaller projects. However, if you are working in a bigger project then I would highly recommend you to use full fledged IDE or integrated development environment. So, why should you use an IDE in a data analysis project? Well, a good IDE can give you a better code writing experience. You can use the features like syntax highlighting, auto code completion, smart indentation to write your codes easily, cleanly, and effectively. In an IDE, you have multiple windows at your disposal, where you can do multiple work and manage your codes easy. Each developer know the pain of debugging the codes. And if you're working in a bigger project with so many scripts, then this pain can aggregate very quickly. So an IDE with good debugging experience can help you to do away with this pain. And when you have so many files, then managing code can also be a tedious task. So an IDE with good version control system can simplify this process for you. For working in R base project, there are plenty of options available to you. And you can choose your IDE as per your conveniences and your requirement. I will list out a few options here for you. First option is the well known IDE eclipse, which is famous among Java developers. So if you are already familiar with Eclipse, then you can download the StatET plugin. Which will help you to create an R friendly environment in your Eclipse IDE itself. You can go to this bit. ly link to download this plugin and get more information. Architect from Open Analytics Group is another option which is an R oriented version of Eclipse. It already comes with a StatET plugin. So you can go ahead with this IDE also. Revolution-R from Revolution Analytics is an enterprise level commercial solution. It has support for big data analytics also. To know more about this product you can visit this bit. ly link. Live-R is another product which provides R based analytics on web itself. Here's the link for more information. So far I have shown you many options. Every IDE has its own advantage and selection of IDE totally depends on you. However, in this course I'll be working on another IDE. That is RStudio. It is also a R oriented IDE. And it is very powerful also. There is an open source flavor of RStudio available to you which is completely free. And you can install it on Windows, Linux, and Mac, all kind of platforms. And it will provide you very similar user experience. This is the reason I'm using this IDE for building these codes. But you can go

ahead and follow along on your choice of IDE also. If you also want to use RStudio IDE, then in the next few clips, I'll show you the installation and a brief overview of RStudio.

Install RStudio on Windows

Installing RStudio Desktop on your Windows machine is very easy. To download RStudio Desktop you can go to rstudio. com, and then you can click Products and then you navigate to RStudio link. This will open the RStudio product page. Let's scroll down a little bit. RStudio Desktop is available in open source and commercial flavor. In this course, we'll be using its open source flavor, which is completely free. So to download the setup, you can click this link. So click it. This will take you to this page. On this page you'll find installers for different operating systems. Windows user can click this Windows link. You can save this file. Depending upon your internet connectivity it will take a while to download this file. I have already downloaded this file. Now to install RStudio to the Desktop all you have to do is to double click on this installer file. Click Next. We will use all the default settings for this installation. So, click Next again. Click Install. So our installation is completed. So now RStudio is installed on your machine. Now to use RStudio you can search for RStudio application on your machine. Here in Windows 8 we can use the charm to search our studio. Type RStudio. Here we have got the search result for RStudio, let's click it. This will launch RStudio. So now RStudio is up and running on your machine. Next we will look at the overview of RStudio IDE in an upcoming clip. In the next two clips, I'll go through the installation of RStudio on Linux and Mac machines. So if you are a Windows user, you can directly jump to the RStudio overview clip.

Install RStudio on Linux

Installing RStudio Desktop on your Linux machine is pretty straightforward. To download RStudio Desktop, you can go to rstudio. com, then you can click Products, and then navigate to RStudio link. This will open the RStudio product page. Let's scroll down a little bit. RStudio Desktop is available in open source and commercial flavor. In this course we'll be using it's open source flavor which is completely free. So to download the setup you can click this link. On this page you will find installers for different operating systems. I have Ubuntu flavor of Linux in my machine so I will click this Ubuntu download link. We will save this file. Now depending upon your internet connectivity it will take a while to download this file. I have already downloaded this file. Here is our file. Next, we can use this to install our studio on our Ubuntu machine. You can install RStudio from terminal also, but here we will be using Ubuntu software center for installing our studio. This

will make installation of RStudio even more simpler. Now just do a double click on this downloaded DB file. This will launch the Ubuntu software center. Now click the install button, to install RStudio on your machine. This will prompt for authentication password, enter the password. Click Authenticate. After this the installation process will resume. And after some time RStudio will be installed on your machine. Now, to launch RStudio we can come up to dash home and search for RStudio. Here we have our RStudio and I'll just click it to launch this RStudio application. So now our studio is up and running on your UBUNTU machine. Now we will look at the overview RStudio IDE in an upcoming clip. In the next clip, I'll go through the installation of RStudio on Mac machine. So if you are a Linux user, you can directly jump to RStudio overview clip.

Install RStudio on Mac

Installing RStudio Desktop on your Mac machine is very simple. To download RStudio Desktop you can go to rstudio. com, then you can click Products and then navigate to the RStudio link. This will open the RStudio product page. Let's scroll down a little bit. RStudio Desktop is available in open source and commercial flavor. In this course, we'll be using its open source flavor, which is completely free. So to download the set up, you can click this link. On this page you will find installers for different operating systems. As we have Mac operating system here, therefore you can click the link for Mac OS here. Here we can get the DMG or disk image file. You can save this file. Depending upon on your internet connectivity it will take a while to download this file. So now our file is downloaded. Since this is a DMG, or disc image file, so to launch RStudio all you have to do is to simply double click this DMG file. This will mount this DMG file on your machine, and you can access it's contents. So here is our RStudio Desktop application. So clicking this RStudio icon here will launch RStudio for you. Open. So now RStudio is up and running on your machine. Now we will look at the overview of RStudio IDE in the next clip.

RStudio Desktop Overview

In this clip I'll give a brief overview of RStudio Desktop. We have already seen the installation of RStudio in Windows, Linux, and Mac operating systems. RStudio look and feel is more or less same in all operating systems. However, from now onwards, I'll be using RStudio in Windows environment in this course. But users in other operating systems can also follow along. Here in this RStudio overview clip, I will not be covering each feature and option available in RStudio. Rather, I will be showing you only the basic features, which we will be using in this course. If you

want to know more about RStudio there is a great documentation available, you can go to Help menu and click RStudio Docs. This will open the documentation page for RStudio in your default browser. Now coming back to RStudio application, here you can see different windows. I'll discuss each window one by one, but let me bring one more window which is to write our code. So to create a new script in RStudio you can go to File menu then New File and then R Script. This will provide one more window where you can write your R scripts. Alternatively you can use this small icon here. You can click this small icon and then you can select R Script option to create a new R Script. So here you can see our whole RStudio screen is divided into four main sections, also known as quadrants. This upper left hand section contains one or more scripts in different tabs. Then on the bottom left section we have our console window. If you recall from our R installation clip we had similar information in this window. This is because RStudio is nothing but the wrapper around the R framework. Therefore it is important to have R installed on your machine before you can use RStudio. If you want to clear this console, you can use Ctrl + L, or Cmd + L in Mac OS. In this upper right pane, we have Enrollment and Viewers tab. In the environment, we can look at different variables, available in different environments or work spaces. I'll talk about environment in detail later in this course. The View tab will be used to display local help contents. On the bottom right section, we have different tabs for different purposes. History tab contains all the commands we have used in your RStudio sessions. In the File tab, you can navigate through different folders and files on your machine. In Plot tab, all put plots will be displayed. In Packages tab, you can see different R packages. You will learn more about packages in a dedicated module later in this course, but I want to give you a brief idea about packages here itself. You can think packages as a group of functions, data sets, compiled codes, or documentations. Each package is developed for some particular set of tasks. As I mentioned earlier also there are over 5, 000 R packages available as of now. In fact when you install this R version, then some packages also get installed. This tab here will show you all the installed package on your machine. And if you install other packages later, then that package will also be listed out here. Here you can see different installed packages, with their package name, description, and version. These check boxes here, represent where that package is attached to the current R session or not. So if you want to use any function in any package, the that package has to be attached. We will cover this whole process in packages module later in this course. So this was a brief introduction to packages. Coming here, in the help tab, different help information will be rendered. We will look at different tabs as and when they will be used in this course. This is the default layout when you install our studio on your machine, but you can change it as per your need. So if you want to change the location or order of these panes, then you can go to Tools, Global Options, Pane Layout, and you can configure the pane you want in the respective

quadrants. Not only this. If you want to change the appearance of your R studio, then you can click this Appearance section. And you can set the font size for the application. And you can also choose different colored themes for your script editor. Let's close it. You can also adjust the size of each quadrant using splitters. Like this. You can also minimize or maximize the quadrants using these buttons. I'll be changing the layouts of these panes, in this course depending upon the requirement. I'm telling you this, so that later you do not get confused when you see the different layout in different demos. So now coming back to our new script window you can write your R script here. Let's put some code here, which you saw in the Hello World demo. As you can see here right of A we are getting some benefit from this R studio editor. On the left, we have the line numbers, so that we can be sure on which line number we are working. We can also see different colors for different code portions, due to syntax highlighting features of R studio. The code shown in the green here, are actually comments. So to put comments, you can use a hash symbol. So in any line, anything after the hash symbol will be treated as comment. Now if you put your cursor on any line such as on this line number 2, and if you want to execute this line, then either you can click this Run button, or you can use your keyboard. You can use CTRL+Enter on Windows or CMD+Enter on MAC to execute this line. Let's do it now. As you can see here, this line number two got executed in the console. Also you can see that your cursor moved to the next line. So, now you can execute this line number 3 also, in the same manner. But if you want to execute this whole script, we can use this Source button here. Here you have two options, either to just Source the script, or to source with Echo. Source with Echo is used to source the script with echo parameter value set to true. If you recall from the previous clip of Hello World, we use the source command with echo set to true to source our script file. So let's do it here also. Let's click on the Source with Echo option. So as we can see here, our whole script was source and executed on the console. One more thing you might have noticed, that as you executed the line number 4, where we have created the variable x, there is an entry in the global environment pane, displayed on this upper right section. Here you can see that the variable name as well as it's type, and it's contents. So in this clip, I tried to give you a brief overview of R studio, you will be learning more features of R studio while going through this course. In the rest of this course we will be using R studio as our IDE in the demos to learn more about R language and R framework. We have almost reached to the end of this module. In the next clip, I'll show you the course structure which will give you an idea how we'll be approaching our learning part for our programming.

Course Structure

Here's the course structure. Primarily, I have divided this course in three main sections. In the next module, which is our first section, I'll show you ways to get help. I will provide you various sources, and if you have just started learning the R programming this will ease your path. After that in the second section which will encompass next few modules I'll talk about fundamental concepts of Core R programming, and R framework. This will help you to build a solid foundation. Finally in the last section you will learn to work with data and do some basic data analysis. So let's dig deeper into the R world. See you in the next module.

Getting Help for R

Introduction

Hi this is Abhishek Kumar and welcome to the second module of the course on R programming fundamentals. This module is all about getting help for R. I have deliberately put this module ahead of core programming modules. Well, when we go to a new place or to an unknown territory we must know ways to get help. Such as to whom we should contact in the case of emergency. Similarity, I wanted to discuss ways to get help if you come across any confusion or problem while learning R programming. By the end of this module you will know where and how to get help if you run across any issue in your R endeavors.

Outline

Here's the module outline. As R framework installed on your local machine is very well documented and well equipped in itself to guide you. So I'll start with the built-in help of leveling R, then I'll talk about doing web search to get online help. And finally I'll discuss the community support and how to get involved with the R community. I'll also list out few do's and don'ts on posting questions in various online forums. So let's start with the built-in help.

Built-In Help

There are various commands or functions in R which can be used to get built-in help. This will work in offline mode also, as it uses the available resources installed on your local machine. And I'll talk about three main categories of such commands. If you want to get R documentation for any object or function, you should use help commands. There are different ways you can use this

command and I'll talk about them in the next clip. But if you are looking for some demonstrations of certain features I've delivered in different packages then you should use demo commands. There is another command that is vignette command which are used to get additional documentation about different features or functions a level in R package. I'll talk about each category of commands in a subsequent clips. So let's start with the help commands.

Built-In Help: Using Help Commands

Help commands are used to search local R documentation for any R function or object. So if you're not sure about working with any function or object then you can use help commands to get instant help. It will search the installed documentation for your search term and will provide you relevant results. The result will be in this form. This might look overwhelming at the first place but once you are used to this you will greatly benefitted. The resulting documentation is categorized into different sections such as description, usage, arguments, values, references, and possibly a set of examples. You can use help commands in mainly two flavors. So if you know the exact term which you want to search, then you can simply pass the search term as a parameter to the help function and it will return you the correct result. Remember, R functions are case sensitive, so make sure you have used the correct case for the search term. There is a shortcut also to use this function. You can simply put a question mark before the search term and it will provide you the same result. So using these techniques you can search for an exact term, but if you do not know the exact search term then also there is a way out. In such scenarios, you can use help. search function and pass the term in quotation marks as a parameter to this help. search function. You can use double as well as single question mark around your search term. This is a shortcut for this flavor of search also. All you have to do is to put double question mark sign before the search term and this will return the same result to that of the help. search function. Now let's see these help commands in action. So here we are in RStudio. So suppose you want to know more about rnorm function which is used for random number generation with normal distribution. Then you can simply put this rnorm term as a parameter to this help function. Let's execute this. I'm using the keyword shortcut of Ctrl + Enter to execute this command. If you are a Mac user, then you can use Cmd +Enter. Optionally, you can use this Run button here to execute this line. As you can see there, we got our search result in the Help tab. It is a nicely formatted documentation. Now let's look at different sections here. First, there is a brief description about this function. Then it provides information about its usage. You can also look into the details of various arguments used in this function. It also provides some details about this function and what are the things involved in this function. The result can also provide details of some return

values. In some documentation, source is also provided. You can also look into the references. See Also section provides some related links. The documentation can also provide few examples. These examples will give you a good idea about its usage. These examples are mostly reproducible in nature. So you can simply copy and paste these examples and can execute them to see them in action. There is also a neat way to execute these examples. As shown here, in the line number two, you can use the example function to execute the example available in this documentation. So here we have powers to the rnorm as a parameter to the example function. Let's execute this. So as you can see here in the console window the example has been loaded and executed. Here in this case there is a plot also so let's press Enter to get the plots. So we have our plots mentioned in the example shown in the Plots tab on the right hand side. So as you can see here, that example function is a really handy tool to get and execute the examples available in the documentation. Now coming to this line number three, as discussing the slides you can also use the shortcut technique to use the help function using a single question mark. Let's execute this. As you can see we got the same results. So you can use this technique if you know the exact search term. Now let's look at the next method here on line number four which is help. search function. This can be used to get search results, even if you don't know the exact term. Let's use the same rnorm term, and put it in quotation marks, and pass it as a parameter. Let's execute this line. So here you can see on the right hand side we have got a list of help pages wherever this term rnorm has occurred. You can look into different help pages by clicking the hyperlinks on the left hand side. Let's scroll a little bit and here we have our normal distribution page link also. Let's click it, and we arrived at the same help page which we had seen previously. Coming back to the script, you can also use the shortcut technique as shown here in the line number five, for the previous command using to get the same search results. Let's execute this line. And here you can see that we have the same search results on the right hand side. So this was all about help commands. Now let's learn another set of commands which are used to see some demonstrations.

Built-In Help: Using Demo Commands

While help commands are used to get documentation for any R function or object, demo commands are used to see the demonstrations of a certain topic or feature available in our package. It is basically a user friendly interface where you can see the running demonstration of a script to understand a certain topic. These demonstrations are focused toward sample usage, and they can make use of some dummy data. The result of these demo commands are mainly code snippets or some related items such as plots or images. Let's see it in action. While help function

is available for nearly all R functions or R objects, demos are not available for all function features. So if you want to see what are the available demos, you can use the demo command without any parameters, as shown here in the line number one. This will list out all demos in all attached packages which can be demonstrated. Let's execute this line of code. As you can see, another tab has been opened in RStudio. And this tab contains all demos available in all attached packages. It means that demo function will show you demos available in all packages for which these check boxes here are checked. Coming back to demo's results, here you can see the demos are grouped together package-wise. Typically we use the term topic to refer to each demo. So the base package contains four topics, while the graphics package has currently six topics. This is not a big list, because we have installed only the base R framework which comes with only essential packages. If you install a package later and if that package is attached and there are demos available in that package, then all demos of that package will also be installed here. Now if you want to see the list of demos in all installed packages, whether they are attached or not, then you can use this command here on line number two. Let's execute it. Here you can see we have got few more demos available in the latest package. If you see here in the Package tab even though this package is not attached then also we caught the most available in this latest package. Now if you want a list out demos specific to any package only, then you can pass the package name to the package parameter of the demo function as shown here in the line number three. Here we want to get all demos in the graphics package only. Let's execute this code. Again we got our result in a separate tab. Here we have all demos available in the graphics package only. It has six topics. Now if you want to see this graphics topic here, which is to show some of the R graphics capabilities, then you can pass a topic name as a parameter to demo function. Optionally you can mention the package name also as a parameter. So now let's execute this line number four. In this demo the console will prompt to hit Enter key to start the demo. Press Enter key. Let's press Enter again to see the plot. Here we have our plot in the plot section. You can again hit Enter to see the next plot. As you can see this demo has different plots so you can hit Enter and see different types of graphics capabilities of R framework. So as you can see, the demos provide full demonstration of a certain topic. So demos are feature specific. But if you want to know more about the package as a whole, then vignettes can be a very useful tool. We will discuss vignettes in the next clip.

Built-In Help: Using Vignettes

So far, we have looked at help and demo commands to get help. While help commands are focused towards providing help for any R function or object, demo commands, on the other hand,

are focused towards any particular topic or feature of a particular package. However, R provides another help mechanism which are known as vignettes. These vignettes are typically short to mid-sized documents which can cover part or all functionalities of a package. And typically this documentation is available in PDF format. So the document will give you a good insight about that topic or package. These documents typically contain brief introduction, background information, usage, results, references, and other related things. This could be very handy if you are new to some package and want to dig down to know more about it's working. Now let's see how we can get these vignettes in R. Vignette commands are very similar to demo commands. So to list out all vignettes in all attached packages, you simply use vignette function without any parameters. As shown here in the line number one. Let's execute this line number one. This will launch another tab in RStudio where all available topics for all attached packages will be listed out. So we have topic name, a brief introduction of that topic, and the format of available vignette. As you can see here, most of the vignettes are in PDF format. Here different topics are group together package wise. So have few topics in the grid package, few topics in matrix package. We have only one topic in the parallel package. Similarly we have topics in other different packages. Topics which are listed out here are from only the attached packages. However, if you want to list out vignettes from all available packages, you can use this command here at line number two, this will list out topics from all installed packages. Let's execute this line number two. Here again, the list of all available topics will be shown in a separate tab. Here in this case you might not find any extra entry with respect to previous command execution as other packages which are not attached in this current R session don't have any available vignettes. But later if you install more packages and vignettes are available in those packages then topics from those packages will also be listed out here. So in this manner you can get all available topics. However, you can also get topics lists for a specific package also. So if you want to get topics from only parallel package, then you can pass that package name as a parameter value for the package argument in the vignette function, as shown here in the line number three. Let's execute this line. Here we got our topics in the parallel package only. Here we have only entry with the topic name of parallel. So if you want to see the vignette for this particular topic. Then you can pass the topic name as the first parameter wrapped into codes, to this vignette function. Optionally you can also specify the package name. Here in this particular case, topic name and package name are same. However, that is not necessary all the time. So, now let's execute this line number four. This will display the vignette in a preview format in your default. pdf viewer. Let's have a brief look at this particular 13 page document. Here you will find an introduction to parallel package. It's working, some concerts, as well as some code snippets, package installation procedures, and there are some references also. So as you can see, this vignette contains all the

information needed to get you going to use this package. These vignettes can be very helpful and handy if you are new to some package. So far, we have looked at techniques and commands to get built in help using your own local machine and for which you do not need any internet connectivity. However, sometimes you may not get all the required information using built in help mechanisms. So we can use web search and do scenarios. Needless to say that for such cases, you will need internet connectivity. So we will talk about web search in the next clip and how can you use it.

Web Search

So far we have looked help mechanisms which can work on information already available and install on your local machine. But if you're not able to find required information from your local machine, then you can use the power of web to find out that information. I'll talk about two flavors of web search in this clip. First, is doing search from your R console itself. There are some R commands which can do the web search for key terms or functions right from your console. However, this method will provide you limited search uses. So you can go to the next flavor of web search where you can use various popular search engines, to get the required information. Now let's talk about the web search from your R console. To invoke a website from your R console, there is R command known as RSiteSearch. It basically uses the search engine of R project available at this link. This function can search for the keywords or phrases in various help pages, vignettes, and task views. Here is a typical search result page. This method can be very useful if you're not getting appropriate help from your built-in help mechanisms which we had discussed in the first section of this module. So let's take a demo to see the usage of this function. Suppose you want to get help on arithmetic mean. Then you can simply pass this key tone as an argument to the all sites search function. We can wrap the search tone in codes, as shown here on the line number one. Now let's execute this line number one. This will launch your default browser and results will be displayed there. Here you can see, we have around 900 documents containing the key terms arithmetic or mean or both. The key terms will be highlighted in red. You can filter the search result, also, you can choose your search target such as functions, vignettes, or task views. So if you choose only functions and then hit the search button again. Then it will list out the search results with search target of functions only. You can also sort the search result as per you need. By default, the sorting will be done using score values. So here you can see, we have some score value associated with each search result. This score typically signifies total number of occurrence of key terms in the search result. One important thing to note here, is that the search result can contain all or one of the search term in any order. But if you want to have an

exact search with both arithmetic and mean term occurring together, then you have put your search term in curly braces as shown here in the line number two. Now let's execute this line number two to see the results. Here our search result count reduced to 256 and if you watch the search results closely, each search result contains arithmetic mean term acting together and they're highlighted in red. These search results may be quite overwhelming and were worse in the first place. That's why I want to show you one more function which can actually group together these search results. And can display them in a tabular and compact form. However, that function is not available in base R packages. The function I'm talking about is available through a package named as SOS. And because that package is currently not installed on our machine, we have to install that package. I talk about packages and their installation process in detail in a separate module later in this course. However, gor the time being, you can just follow along with me. So to install any package, we can use install. packages function and then we can pass the desired package name as an augment. Let's execute this line number three. This will download and install the SOS package on your machine. Next, we have to load that library into the memory before we can use any function of the SOS package. This is a typical process to use a function after installing any package. So here we have used the library function and passed to the package name as an argument. Let execute slide number four. So now that library is loaded, we can invoke the function availabe in this SOS package. That function name is findFn. As shown here on the line number five. This function takes a search term as an argument, here also for exact search, you can wrap your search term in bases. Let's execute it now. The final search result will be rendered in your default browser. As you can see, the search results are organized in a much better fashion here than the R site search function response. However, the result from both the functions will be seen. In fact, internally, this findFn function uses the R site search method only. This findFn function does the job of subsetting and formatting of the search results coming from the R site search method. So, as you can see, we have a table structure for our search results. Let's quickly look at few important columns. Here on the last column we have links to the actual search help page. Package column shows the name of the package name containing the help page. Function column on the other hand contains the help page name itself. Count column signifies a total number of matches in the current package. While the score column denotes the score value coming from the our site search function output. The max score column here contains the maximum of the score value of the various help pages in the corresponding package. Means if the package has ten help pages satisfying those search criteria, then the maximum of the ten scores will be reflected here. Total score column signifies some of the all scores for the particular package. Moreover, as you can see here, there were ten pages of results. So if you want to reduce the number of results, or you want to see only the top pages, then you can set the max pages

parameter as shown on the line number six. Here we have set the max pages to two. So it will result back only top two pages. Let's execute this line number six. Here we got a lesser number of search results in top two pages only. As we had set the max pages value to two. This is also a short cut of using this findFn function as shown here on the line number seven. You can use three question marks followed by the search term in quotes. You can also add the max pages value in the parenthesis. So this shortcut here on the line number seven will do the same job to that of the previous line number six. Let's execute this line number seven, and again we get the same result back with two pages. So in this clip we looked at the web search techniques which can help you to perform such searches right from your console. However, these techniques provide limited search results because these commands are using the our project search engine only. But you can also use the power of various popular search engines to perform various kind of searches. You must be using your favorite search engines in your daily work. And these search engines could be very handy in finding solutions of variety of issues that you may encounter by learning or working with our language or our framework. You're already familiar with using search engines, but I would like to give you a couple of tips which can help you in getting the right information easily. Such as, using R in a square brackets instead of simple R while performing searches, may give you better results in search engines. And if you have encountered any error, then you can directly use the main error message as your searchdom and you may get search results which can help you to trace and resolve that error. So far we have looked at various help mechanisms, such as built-in help, using various commands, or web search using various search engines. But sometimes, you may encounter such situations where you are not able to find any kind of help information. In those scenarios, you can take the benefit of various R communities. Where you can post your queries and get help. So I'll talk about communities in the next clip, where we will look at various kinds of available community support. I'll also list out a few do's and don'ts while interacting in the community

Community Support

One of the most compelling features of using the R framework for your data analysis projects is the active community support. R has a great community support where you can post your problems and get your solutions back. And there are a number of ways you can interact with the community. You can join various mailing lists dedicated toward all related work. Or you can post your problems or issues in various online forums. You can also look at various blogs maintained by many passionate individuals and groups. These communities are not only to get help, but you could also contribute your knowledge and share your experience with the community. These

infractions will help you to further solidify your own understanding. So now let's know more about these communities starting with the mailing list in the next clip.

Community Support: Mailing List

Mailing lists can be very useful to get various R related information and notifications at regular intervals. So you can subscribe to various mailing lists, and information will be available to you in your mailbox. You can even post your problems in these mailing lists. And people are more than willing to help you our if your problem is genuine, and you stick to certain protocols while asking questions. There are several mailing lists dedicated towards R related development and activities. However, two mailing lists I want to specifically to mention here. First is the R help mailing list, which is a general mailing list about R related work. You can post your questions here and get solutions. R devel is another mailing list which is more focused towards code development in R. So while R help is a generic mailing list, R devel mailing list on the other hand, is for more technical stuffs. Not only this, there are several mailing lists for special interests groups for people working in specific areas. So if you are using R in the financial sector, you can join R-sig-finance mailing list. Similarly for users working in high performance computing area, can join R-sig-hpc mailing list. There are several other special interest groups also. So, to know more about various R-mailing lists, or to subscribe to these mailing lists, you can visit this bit. ly link here.

Community Support: Forums

Forums are, again, very helpful platforms if you got stuck to any problem or if you want some guidance in using R. There are several online forums dedicated towards R. Stack Overflow is a popular online forum among developers working in different programming languages or frameworks. Here is the bit. ly link for the Stack Overflow topics tagged with R. Cross Validated is another forum which is a sibling of Stack Overflow forum. This particular forum is focused towards statistics and determining related questions, but mainly, using the R framework. Another very interesting forum I want to mention here is Nabble. It is a very active online forum as well. I have included bit. ly links for all these forums so you can take help from these forums and can provide your contributions as well.

Community Support: Blogs

Blogs are great platforms to share knowledge and experience and blogs can be very helpful to you also if you are looking for some information that is already mentioned and explained by

bloggers across the globe. There are several blogs sites on R, however, I want to mention few of my favorites here. First is, R-Bloggers, which is a mash up of several R related blogs. So, this is a great resource depository. Blog by Revolution Analytics group is another well maintained blog site. And you can get plenty of information there. If you are interested in statistics, then R-Statistics is a great blog to follow. Similarly, there is another very useful blog on the R-Data mining area. I have included bit. ly links for all blogs. These blogs are regularly maintained and updated by very passionate individuals and groups. So, you can view them, subscribe to them and get benefits from them.

How to Ask a Question?

So far I have discussed various kinds of community support platforms such as mailing list, forums and blogs where you can post your questions or issues. However, if you want to make the most of it, you should follow some basic protocols. And if you follow these ground rules then you can get your answers quickly and precisely. Let's look at some of the best practices you should follow while posting an equation in various forums. If you come across any issue or error, then it would be great if you can make a minimum reproducible example. This example can make use of some dummy data. Moreover, you should try to reproduce the error in the minimum lines of code. These examples will help others to reproduce the same error in their own machines easily, so that they can quickly give you suggestions to resolve the issue. So while posting your questions, you can also mention the expected results, as well as the results you're obtaining. Other than that, few necessary information would be great. Such as, your R version, or version of the specific package you are using. You should also mention the operating system. This information is required because there could be some bug or issue specific to some R version, or package version, or the operating system itself. If there are some additional information available which can help others to trace back the error, then you should mention them as well. So overall, the whole objective of providing all required information in one go is to pinpoint the exact problem or issue in the minimum number of iterations and in minimum time frame. If you want to know more about posting guidelines then you can visit this bit. ly link. All of the guidelines available on this link are specific for our mailing list, but you can follow them to ask questions on other forums as well. Here are a few do's and don'ts while interacting in various art communities. Let's look at the do's list first. Good manners are obviously on the top of the list in any community interactions. Then you should always do your homework before posting any questions, or before making any comments. So far, I have shown you different forums, or mailing lists. Each forum and mailing list has a certain objective and certain focus area. Therefore, you should post your credits in the

correct mailing list or forum only. As I mentioned in the previous slide, you should always provide minimum required information about any issue or problem. So these are a few things we should always follow and take care. Now let's see if few things you should avoid in any community infractions. Your questions should always have a subject or objective. So make sure your subject headers are properly written. Another common mistake which people tend to make is that if they are not able to get expected results at the first place, they claim to have found bugs right away. But there may be chances that you are using the older versions, and those bugs have already been resolved in the future releases. And if you are using some functions and you're getting odd results, then don't start judging the function itself. There may be chances that you are not using those functions correctly or in the manner they are intended to be used. You should also avoid threatening of not to use the software just because your not getting required help or answers. I mention these several do's and don'ts here so that you get your answers and clarifications easily if you are stuck to some problem. But even if you are not getting answers, don't be disheartened. I'm sure you will get the right information eventually.

Summary

So, in this module, we've looked at three broad categories of getting help for R. First of all is built-in help, which is available to you using the local resources installed on your machine. We look at three set of commands, first was the help command which can be used to get the documentation for any R object or function. We discussed two flavors of using help commands. So you can use help function if you know the exact search term, and you can use help dot search function if you do not know the exact search term. We also learned the use of demo commands to see the live demonstrations of various R features. Then we looked at vignettes, which can be very helpful if you are new to some package. These vignettes can give you all information required to get coined with any package or topic. After looking at the built in help, we discussed Web Search, where we first learned the use RSiteSearch command using which we can do a web search from our R console itself. Then we looked at SOS package, in which the FindFn function can be used to format the search results in a tabular form. Then we disclosed how various popular search engines, such as Google or Bing, can help you to get the required information quickly. Finally, we discussed the third broad category, which was community support, where we looked at various mailing lists, forums, and blogs. Where you can post your questions and get back the solutions. You can also share your experience and can be an active member of these communities. I also mentioned a few do's and don'ts which you should take care of while doing any community interactions. Hopefully the resources provided in this module will ease your part in learning R

programming in R framework. Now, for the next module, we will dive into the core R programming topics where we will try to build a solid foundation on R framework. So, see you in the next module.

R - Variables and Operators

Introduction

Hi. This is Abhishek Kumar, and welcome to the third module on R Programming Fundamentals. This module is also the first module of Core Programming section. In this module, we will discuss some of the fundamental topics in R Programming, such as variables and operators. A good understanding of these topics will help you to understand the nitty gritty of our language. This module is full of demos so that you can follow along with me and can further solidify your understanding on these topics.

Outline

Here's the module outline. We will first talk about variables. We will look at a few naming conventions as well as Google style guide to declare variables appropriately. We will also learn about our environments in the context of variables. Then we discuss few basic R operators such as arithmetic operators and logical operators. And finally we'll discuss the concept of vectorized operation that makes R a very powerful language so let's start with variables in the next clip.

Variable

A variable can be considered as a placeholder or container which can hold any object. For example, here x is a placeholder which can hold a number 10. So to assign a value to a variable, we can use the assignment operator which we saw in the introductory module. The assignment operator is denoted by less than symbol followed by a dash symbol. Also note that we have used space before and after the assignment operator in the statement. This spacing is not mandatory, but ideally you should use such spacing as it increases the programs readability, and reduces any unwanted confusion. Another important point to remember that R variable names are case sensitive in nature. It means the variable x in lowercase is different from a variable X in uppercase.

Naming Convention

Let's look at a few naming conventions forcreating variables with syntactically valid names. so, a valid R variable name can consist of letters, numbers, dots, and underline characters. Moreover, a syntactically valid name can either start with a letter or a dot character, provided a number does not follow the dot character. Let's look at few syntactically valid and invalid variable names. So these variable names here on the left are all valid variable names. Top three names are valid because they all contain letters, dot or underline character only. And all of them starts with the letter. The last variable name is also valid, because even though it starts with a dot character, but a number does not follow it. Let's look at this invalid variable name. This variable name is syntactically invalid because, although it started with a dot character, but dot is followed by a number. Moreover, you cannot use few reserve keywords to name your variable. Here are those reserve keywords. These keywords have some special meaning, and some semantic definition in the art programming language. Therefore, you should not use these keywords to name your variables.

Naming Guide

So far we have discussed rules for creating syntactical valid variable names. But should we use any valid name to name our variables? Well the answer is no. Although the framework will not give you any error as long as you are using any syntactically valid name, for your variables. But it is often a good practice that you follow some good naming guide to ensure uniformity in your code. Especially if you are working on a bigger project in a team environment. If every team member follows a proper coding standard, then it will be very helpful for others to review, or to work, on another team member's code. You can always formulate a coding standard for your own project. However Google R style guide is a great standard to follow in your R related work. Here is the bit. ly link for this guide. This guide will provide you style guidance for several coding practices including variable names. So as for the Google R style guide, the variable name should contain only lowercase letters. You can use dot character to separate different words of the variable name and you should not use underscore or hyphens in your variable name. Now let's look at few variable names. This first variable name can be turned bad, because it contains an underscore character. Second variable name can be considered just okay, because although it does not contain any underscore, but it contains upper case letters, and moreover, words are not well separated. Now look at this third variable name here. It can be considered good, because it does not contain any underscore, and it has only lowercase letters. And moreover vaults are also properly separated using dot character. So this was about variables naming guide. So far we have

discussed radius naming conventions and naming guides. Hopefully this will help you to create not only syntactically valid variables names, but also consistent and more readable variables names.

Assign Variable

So far we have discussed rules and guidance to name a variable. Now let's see how to assign a variable. We have already looked at one technique to assign some value to a variable, in which we can use the assignment operator like this. However there is another way you can accomplish this assignment task. You can use assign function to do so. I'll talk about functions later but you can think functions as a box which can take some values and perform some actions. The assign function takes several parameters however, we will be using only two parameters here. First parameter is the variable name as string, and second parameter is the value that we want to assign to the variable. Now lets look at both techniques in our studio. Here on line number one we have created a variable match. score to store the score of a cricket match inning. The variable name here follows the Google R style guide, as it has all lower case letters, and we have used dot to separate the words match and score. Now suppose the score created in a cricket match inning is 300, so here we have used the assignment operator to assign a value of 300 to the variable match. score. Let's execute this line number 1. So here on the console you can see the line number one has been executed and we have assigned a value to the variable match. score. Now if you want to see the content of the variable match. score then you can simply execute the variable name as shown here on the line number 2. Let's execute this line. Here the value of match. score variable got printed on the console. You can also use assign function for assigning a value to the variable action here on the line number 3. We are passed the variable name, match. score, in codes as a first parameter. Remember, in R you can use single as well as double quotes interchangeably in most of the places. So whenever I use the term quotes it means that you can use either of single or double quotes until or unless it is explicitly mentioned. So this is our first parameter to the assign function. Then we have passed the value of 300 as the second parameter. Let's execute this line. So our statement has been executed so again if you want to print the content of this variable you can simply use the variable name to do so as shown here on the line number 4. Let's execute this line. And here again we get to 300 as the result. So this as about assigning values to a variable. Another important thing to notice here that whenever you create variable then that variable will also be included in the environment tab here. Environment is a very important concept in R because they can help you to understand various intricacies of R programming. So let's talk about the basics of environment in the next clip.

Environments and Variables

Environment can be considered as a bag or a place to store variables. These environments are used in lexical scoping. Scoping is an advanced topic which we will cover in an upcoming course. However, I want to give you a brief introduction to environments in the context of variables here in this module itself. So when you create variables such as x and y from the console, then by default you are creating those variables in the global environment. But you can create variables in a custom environment also. Here we have a custom environment, inside the global environment. Means global environment, is the parent of the custom environment. As you can see here we have x and y variables in both custom as well as global environment. And the same variable has different values in different environments. Means a variable named as x can have a value of 300 in the global environment and another variable with the same name x can co-exist in a custom environment with altogether different value such as 320. To create a new environment in R, you can use new. env command. Then if you want to create a variable in the custom environment, then you can do it in many ways. You can use the assign function, which we looked at in the previous demo. Here you can pass the environment as a parameter other than the variable name and its value. Alternatively, you can use a variable name wrapped in code, in a double square bracket as shown here. So you can read this line as assign the value of ten to the variable x in the environment named as my. environment. You can also use a dollar sign followed by the variable name, to accomplish the same task as shown here. And if you use this dollar sign technique, then codes around the variable name is optional. So, this was about creating or assigning some value to a variable in a custom environment. But if you want to get a variable from any environment, then, you can use the get command. You can pass the variable name as well as the environment as the parameters to get the variable. You can also use double square bracket to get the variable back. The variable name should be wrapped in quotes. Alternatively, you can use dollar sign followed by the variable name to get the variable from any environment. And again if you use this dollar sign technique, then quotes around the variable name is optional. Now let's look at these commands in action. Here on line number two we are creating a variable, match. score, and setting its value to 300. Remember, if you execute this line then it will directly executed on the console. Therefore, the variable will belong to the global environment. So next execute this line. Now if you want to see the value of this variable, you can simply use the variable name itself as shown here on the line number three, means if you directly execute any variable name on the console, the console will display its contents. So let's execute this line. So here we got the value of 300. Now let's make sure that we created the variable in the global environment. For this we will use get command as shown here on the line number 4. We can pass the match. score rapped in

codes which is the variable name as the first parameter. So second parameter of this get function is the environment itself. To get the global environment we can use globalenv function. This globalenv function will return the global environment. So here we are trying to get the value of a variable match. score from the global environment. Let's execute this line. And here you can see on the console, we get the value of 300 as a result. So we are now sure that the variable we created on the console belonged to the global environment. Now let's create a custom environment. We have used new. env command to create a new custom environment named as my. environment. Let's execute this line. So our custom environment is created. Now let's check the parent environment to this newly created custom environment. There is R function to get the parent environment. We can use parent to. env function and pass the environment to get its parent environment. So here we have passed a newly created custom environment my. environment as a parameter. Let's execute and watch its result. As you can see we got the global environment as a result. So typically default parent provided by the new. env function. Is the environment from which it is called. And because we have used this new. env command from the global environment. That's why we are getting the parent environment as the global environment. Now, to assign a variable in a custom environment, we can use any of these techniques shown on the line number 11, 12, or 13. Each line will do the same job. On line number 11 we have used assign function. While on line number 12 we have used double square brackets. And on line number 13 we have used a dollar sign. Moreover, in all these three lines, we have deliberately set the value of the variable, match. score, to 320. Let's execute these three lines, 11, 12 and 13. To execute all these three lines, we can select these three lines, and click the run button here. So all these three lines got executed in console and every line will do the same task. Now, if you want to see the contents of the variable, match. score created in customer environment, then you could use any of these three lines 16, 17 or 18. Online number 16 we used get command, and passed variable name and custom environment name as parameters. While on line 17, we have used double square brackets and on line number 18, we have used dollar sign followed by the variable name. Let's execute all these three lines at once. Here on console you can see that regard the value of 320 instead of 300, in all the cases. Because we are extracting the valuable match. score from the custom environment named as my darting environment. So as you can see the variables with same names can coexist in different environments. So this was a brief introduction to environments in the context of variables. Now we will look at another important topic in our programming that is Operators.

Operators

Operators are one of the pillar blocks in any programming language. Typically an Operator can work on one or more objects, also known as Operands, and it will return some of the results. Well, so far in this course e have used various functions and primarily job of a function is also very similar means. To take some input parameters, perform some task, and possibly return some results. So you can think these Operators act just like functions, however, both are syntactically different. In our program language, mostly you will encounter two types of Operators. First is an arithmetic operator which typically operates on numeric values. And second is a logical operator which typically operates on Boolean or logical values such as true or false. We will discuss both types of operators one-by-one starting with arithmetic operators in the next clip.

Arithmetic Operators

There are several arithmetic operators in R and they will look very familiar if you have worked in any other programming language. So we have addition, subtraction, multiplication, division. Then we have exponentiation operator shown as caret sign. These another way of using exponentiation operator. That is using double multiplication signs. R has modulus operator also, shown by two percentage signs. This is to get the remainder value, if you divide one number by another number. Another arithmetic operator is integer division. Shown by the division sign in between two percentage signs. As you saw in the previous slide, R has several basic arithmetic operators. Along with that, R has several in-built mathematical functions, also to perform various kind of mathematical calculations. Due to these features you can use R as an advanced calculator also. Now let's look at these arithmetic operators, and some important mathematical functions in action. Here we are in R studio. Let's use our first arithmetic operator that is addition as shown here on the line number two. Here we are adding two numeric values, 10 and 5, using plus symbol as addition operator. Let's execute this line number 2. Here we have the result of 15 on our console window. Let's quickly wrap up, other operators also. We can use dash symbol as subtraction operator, as shown here on the line number 3. Let's find the result of line 3. So we get the value of 5 as a result. For multiplication you can use star symbol as shown here on the line number 4, let's quickly run this line number 4, so we have 50 as a result of this multiplication operation. Similarly you have division operation as shown here on the line number 5 let's execute this line. So we get the value of 2 as a result. Now there are 2 ways for exponentiation. First using caret symbol as shown here on the line number 6. Let's execute this line. So we get the result here on our console. The result is displayed in scientific notation. It means 1 into 10 raised to the power plus 5. But if you do not want to see the result in the scientific notation, then you can use format function, as shown here on the line number 7. So the first parameter is our calculation,

while as the second parameter we have set the value of scientific Boolean parameter to be false. Make sure you write false in all capital letters. Let's execute this line number 7. Here you can see on console, we have our result in regular format not in scientific notation. But if you look closely at the result, the resulting value is wrapped in codes. Because format converts the result in string, and therefore it is used just for pretty formatting. Now moving to the next line in our script, as I mentioned in the slides you can also use double star sign to find the exponent action here, on line number 8. Let's execute this line number 8. Here we get the same result back in scientific notation. Now you know, how to formulate the result in the regular notation. Now let's look at couple of more arithmetic operators. Here on line number 9, we have the used the modulus operator which is used to find the remainder value if you divide 10 by 3. Let's execute this line number 9. So as expected we got the result of one in this case. For integer division you can use respective arithmetic operator as shown here on the line number 10. Lets execute this line, and we got the result of 3 which is the integer part of division result if you divide 10 by 3. Now let's look at some common mathematical functions. First is ABS function as shown here on the line number 13. This function can be used to get the absolute value. Let's execute this line number 13. As a result, we got the absolute value of 5. Next we have used the log function as shown here on the line number 14. This function will return the natural logarithm. Let's execute this line number 14. And we want our result back, but if you want to get the logarithm using 10 as a base then you can pass the value of 10 as a base parameter to this log function. Action here on the line number 15. Let's execute this line, and we got the desired result on the console. There are commands to get exponential and factorial values also, as shown here on the line number 16 and 17 respectively. Next, execute both lines, 16 and 17. So here on the console we have our results. There are some special constants also, such as pi, which is commonly used in many calculations. So you can use pi in small case as shown here in the line number 20 if you want to use this constant value. Let's print its value by executing this line number 20. Here we have the value of pi on the console. But here you can see we have only six digits after the decimal. Because this is the default global setting, in the r framework. Let's check out the global settings. For this you can use the command options. As shown here on the line number 21. Let's execute this line. As you can see there are plenty of global settings displayed on the console. However, if you scroll a little bit. Here you can see the settings, for the the digits parameter. It is by default set to 7. The value 7, means 6 digits after decimal. You can change this setting using the option command itself, as shown here, on the line number 22, where you can set the digits parameter to something else per your requirement. For example, he we have set the digits value to 10. Let's execute this line. So we have set the digits setting to 10. Now let's print a value of pi again. Let's execute this line number 23. Now as you can see, we have the value of pi with nine digits after the decimal. So if you encounter any

issue on precision in your project, you know where to look for that setting. So in this clip we looked at various arithmetic operators and few mathematical functions. I have shown only a few mathematical functions here, but you can look at various other mathematical functions also in the r documentation. But, one thing I want to point out is that, so far we have used various arithmetic operators and mathematical functions with real numbers. And results of all operators and functions we are also all real numbers. But suppose you use the division operator to divide a number such as 1 by 0 in such an area you cannot write the output as a real number. Because such cases are special cases, and to handle them in r there are several special numbers which we will look at in the next clip.

Special Numbers : Inf, NaN, NA

R is well equipped to deal with very special cases, such as division by 0. Or doing calculation that doesn't make any sense. Or performing analysis that involves missing data. There are few special numbers in r, which can be used to tackle these special cases. Let's discuss these special numbers in brief. So the first set of special numbers in r is positive and negative infinity. Denoted by Inf and minus Inf respectively. Typically infinity value represents overflow conditions, or some number that is so big. Or so small that it can not be handled by your computer. Another special number is NaN or not a number. NaN is used to represent a value which can not be represented as a real number. Typically you can come across NaN, if output of any calculation does not make any mathematical sense. The third special number is NA or not available. You can think NA as missing values. You will often encounter NA values, in your data analysis projects. Because in a typical real world scenario some of the measurements, and values will be missing or not available in your data set. You may be wondering why these terms are called special numbers. Well, they are called special numbers because they allow calculations or analysis to continue or to terminate gracefully in any adverse situations. Such as if you hit an overflow condition. Or when a calculation reaches a stage, where it does not make any mathematical sense. Or there is some missing data. If a language does not have such provisions of special numbers, then your program can crash abruptly. And the framework will have no idea what to do next. Now, let's take a demo to know more about these special numbers. If we divide a number such as 1 by 0. Actually on here on the line number 2, then it will desert in positive infinity. Let's execute this line. As you can see, r framework handled this division by 0 quite elegantly, and we have Inf as a result on console. Inf means positive infinity. To get negative infinity, we can divide a negative such as minus 1 by 0 as shown here on the line number 3. Let's execute this line. So, we have minus Inf as a result, which means negative infinity. Moreover, if you would perform an arithmetical operation such as

additional subtraction or multiplication, which involves infinity, then the outcome will also be infinity. Here, on line number 4, we are adding infinity to area number 5. Let's execute this line number 4. So as you can see, the result of the addition is infinity again. In r, there are functions available to check if a calculation gives an infinity result or not. So we have is. finite function, which returns a true value if the expression passed to this function evaluates to a finite value. Here, on line number 5, we have passed 1 divider by 0, as an expression, to is. finite function. Let's execute this line. As expected, we have an output of false, because 1 divided by 0 is infinity. There is is. finite function, also, as shown here on the line number 6, which works, just opposite to is. finite function. So if the expression inside this function evaluates to an infinite value, then this function will return true otherwise, it will return false. Let's execute this line number 6. As expected, we have a result of true on the console. You can use these functions to check your expressions, before doing any kind of processing. So this was about infinity. Now let's look at few samples of NaN values. You can get NaN if you divide infinity by infinity as shown here on the line number 9. Let's execute this line number 9. So we have NaN as a result. Typically r will return NaN if any computation leads to an undefined value. There is r function, is. nan, as shown here on the line number 10 to check if a value or expression evaluates to NaN or not. So if the expression passed through this function, evaluates to some undefined value, then this function will return true, otherwise, it will return false. Let's execute this line. So as we had passed, infinity divided by infinity as an expression to this is. nan function. Therefore we got the result of true in this case. Now let's talk about anys or missing values. Well, any arithmetic operation that involves NA dissolves into NA. Here on line number 13, we have added the value of 5 to NA. Let's see its result. So here, as expected, we got any result on the console. Again, this is a function is. na. To check for NA action here on the line number 14. So you can check an expression for NA. So this function will return true if the expression inside this function evaluates to NA. In this sample we have passed NA itself to this function. And it should return true. Let's confirm this. Execute this line number 14. And as expected we got the output of true. Another important thing to remember that in r NaN is NA but converse is not true. So if you pass NaN to is. na function as shown here on line number 17 then it will return TRUE. However if you pass NA to is. nan function as shown here on line number 18 then it will return false. Let's execute both line number 17 and 18 to confirm. So as you can see for the first function we got a out of TRUE. While for the second function we got an output of FALSE. So this was a short discussion about spacial numbers. In the next clip we will cover another broad category of r operators that are logical operators.

Logical Operators

R has several logical operatives also, and they will be really handy when you will start working in your data analysis projects. Let's quickly look at some logical operators in R language. So we have greater than, greater than equal to, less than, less than equal to, equals to, not equal to. Then we have logical NOT, logical OR, and, finally, logical AND. These logical operators are very easy to use. Let's take a short demo to see all these logical operators in action. Let's quickly go through various logical operators of R language. Here from line number one to seven, we have used some real numbers to demonstrate the use of logical operators by comparing two numeric values. Let's execute line number one to seven in one go. So as you can see on the console, you have Boolean results for each comparison. These results are very obvious so five greater than two is true. Five greater than equal to two is also true. Five less than two is false. Five less than equal to two is also false. Five equals to two is false and five not equal to two is true. However, I want to show you that these logical operators do not only work on numeric values. They can work on character strings also, as shown here on the line number eight. Let's execute this line number eight. Here we got the result of true. However, remember that the result of character string comparison is lexicographic, and can depend on many things such as correlations and encoding. Now let's look at a few more logical operators. So logical NOT operator will simply inverse the value of the logical expression inside the parenthesis. So if we apply the logical NOT operator on the value true, as shown here on the line number nine, then we will get false as a result. Let's confirm it by executing this line number nine. So as expected, we get false as a result. There are other logical operators also, such as logical OR and logical AND. Logical OR works on two logical expressions and if any of two logical expressions evaluates to true then final output will be true. Otherwise the output will be false. Here just for demonstration, we have used logical OR operator between true and false as shown here on the line number ten. Now let's execute this line to see the result. Here on the console, we can see we got true as a result, because as mentioned earlier, if any of the two operands evaluates to true, then the result will be true. Now for logical AND, the result will be true only when both operands or logical expression evaluate to true. So we should get the output of false if we will execute this line number 11 because of one the operands is false. Let's confirm the result. And as you can see here, we got false result as we had expected. So in this clip we covered several logical operators. Now in the next clip we will take another important concept in R language, that is vectorized operation. Vectorized operation make R a very powerful language. So let's find out what do we mean by vectorized operation, and how it is useful to us.

Vectorized Operations

To understand vectorized operation, you first have to understand what is vector. Although I will discuss vector and other data structures in the next module, but I'll give you a glimpse of vectors in this module itself so that you can have a good feel of vectorized operations. Well, you can think vector as one dimensional set of values which are of similar type. Similar means all members of the vector may be numeric in nature or they all may be of character type. Let's take an example of a fictitious class with four students Raj, Rahul, Priya and Poonam. I will use this example in future modules also to clarify various concepts. So suppose all of these students took a classroom test, and they scored these marks respectively. Here all these four marks are of numeric type so you can collectively call these four marks as vector of marks. Now to create such vector in R language, you can use C or combine function. Here we have created a variable named as student. marks and assigned the output of combine function to this variable. In this combine function, we have passed all four marks as arguments, so you can use combine function to create vectors by combining discrete values. Although we have passed numeric values here, but this combine function works on all data types such as character or Boolean. We will see more of its usage in the next module. There are different ways to create a vector and we will look at them as we will progress further in this course. So here the student. marks is our vector, or each of these student marks is a member or element of this vector. So now you have learned about how to create a vector, let's learn about vectorized operations. Vectorized operations typically involve operators or functions which can work on one or more vectors and give some results. The real benefit in R language is that most of the basic functions or operators in R language are vectorized in nature. In R language, you will encounter different flavors of vectorized operations, and we will have a look at some of those flavors one by one.

Types of Vectorized Operations

Let's look at first type of vectorized operation where an R operator or function can work on different elements of a vector, and can produce a scalar result. Scalar means a single value, but the result can be thought of as a vector of length one. Let's take an example of mean function in which we can pass a vector. Suppose a vector contains three elements, E1, E2 and E3. So the mean function will work collectively on all elements of the vector, and will return some reserved value, E. E is nothing but an average of all elements, even E2 and E3. So you can see here a single command can work on all elements of the vector without writing any explicit loop, and thus making code more efficient and fast. So as you can see in this flavor of vectorized operation, the import is a single vector, and the output is a scalar or vector of length 1. Now let's see second type of vectorized operation, where an R operator or function can again work on different

elements of a vector but produce a vector result. Let's look at sqrt function to get square root. Typically in traditional language, this function takes a numeric value as an argument, but in R language, it can take a vector also. Suppose the vector contains three elements, E1, E2, and E3. So the resultant output of sqrt function here will be a vector which contains square root values of all three elements even E2 and E3. So the single command will do the job of finding square roots for all elements of the vector. Again you can see, due to vectorized operations, we have to write small pieces of code without explicit loops, thus making our code more efficient as well as more readable. So in this case, the resulting vector size is same to that of the input vector. Size or length of a vector means number of elements in that vector. So the input vector to this function has a length of three, as it contains three elements. And the output vector of this function has also the size of three, because the resulting vector also contains three elements. So this was about the second type, or second flavor of vectorized operation where the input is a vector, and output is also a vector. Now let's see the third type, or flavor, of vectorized operation where an R operator or function can work on multiple vectors and produce a vector result. Let's look at the example of a simple addition operator. Here we are adding three vectors, V1, V2 and V3. Here each vector contains three elements, means each vector is of size three. So the result of this operation will again be a vector which is also of size three. First element of the resulting vector will be the sum of all first elements of each vector. Similarly, we can find second element of the resulting vector and the third element of the resulting vector. So this was our third type, or flavor, of the vectorized operation where the input is multiple vectors and output is a single vector. And again we are reaping the benefits of vectorized operations here. With a simple line of code we are able to accomplish our task. We do not have to write explicit loops and our code is also more readable. So far we have looked at different flavors of vectorized operations. Now let's see them in action in a demo. Let's first create a vector. Here on the line number one, we are using combined function to create a vector of four numeric values. These numeric values represent marks scored by four students in a fictitious classroom test. We have assigned the vector to the variable student. marks using an assignment operator. Now let's execute this line number one. So a statement has been executed and students. marks variable has been created. Now if you want to see the content of this variable then we can simply execute the variable student. marks on the console as shown here on the line number two. Let's execute this line. As you can see, content of this variable student. marks has been printed on the console. It has four numeric values, 10, 20, 30, and 40. Now let's see a sample vectorized operation of level one where the function, or the operator, can work on a single vector and can return a scalar value. Suppose we want to get average marks for all four students. Then we can use mean function available in R, as shown here on the line number four. This mean function takes the whole vector student. marks as an

argument. Let's execute this line number four. As you can see here on the console, we got average marks, which is a single scalar value. This single value can also be thought of as a vector of length one. Now let's see a couple of vectorized operations of level two where the function or operator can work on a single vector, and returns a single vector. Suppose the teacher who took the classroom test was very happy with the handwriting of all the four students. And he has decided to give five extra marks to each student as bonus marks. So here on the line number six we have used an addition operator to add a value of five to the variable student. marks. We have also saved output of this addition operation back to student. marks variable. Let's execute this line number six. So our statement has been executed. Now let's see the updated content of the variable student. marks. For this, we can again use the variable name as shown here on the line number seven. Therefore, the output of this addition operation on line number six is again a vector of four numbers. Now let's execute this line. So here on the console, we can see that the value of five has been added to all four members of student. marks vector. And the output of the addition is, again, a vector of four numbers. But not only arithmetic operators, you can use logical operators also to work on each member of a vector. Suppose we want to know which student has got the passing marks of 30 in the classroom test. So for this we have used a greater than equal to logical operator, as shown here on the line number eight. Let's execute this line number eight. Let's closely look at the results. Here we have got four Boolean values as a result. So we have true for all those cases for which marks is greater than equal to 30. Here we have two true values and two false values. And these four Boolean values also form a vector. So as you can see again the result of the vectorized operation on a vector is a vector. Now let's look at a sample scenario of the vectorized operation of a third flavor where we can use the operator function on multiple vectors and the result will be a single vector. Suppose all four students of the fictitious classroom took two tests on physics and chemistry. Here on the line number ten we have again used combine function to create numeric vector, student. physics. marks, with four numeric values. And similarly on line number 11 we have created a numeric vector student. chemistry. marks. These two vectors represent marks scored by four students in physics and chemistry subjects respectively. Let's execute these two lines numbering 10 and 11. So we have created two variables, student. physics. marks, and student. chemistry. marks. Now we want to get total marks obtained by all four students, by adding marks obtained in physics and chemistry. Here on the line number 12, we have created a variable, student. total. marks, using an assignment operator. On the right hand side of the assignment operator, we have used addition operator to add two vectors, student. physics. marks and student. chemistry. marks. So here the addition operator is working on two vectors, and the result is a single vector, student. total. marks. Let's execute this line number 12. So the statement has been executed on the console, and the new vector, student.

total. marks, has been created. Now to see the content of this newly created vector we can simply use the vector name as shown here on the line number 13. Let's execute this line, and on the console we can see the content of the vector student. total. marks. So in this clip we saw different flavors of vectorized operations. And we did not have to write any explicit loop to perform these vectorized operations. These vectorized operations will be very handy in your data analysis projects. And we will look at more of its usage in upcoming modules also.

Summary

So in this module, we covered three main aspects of R programming language. We started with variables, where we learned that a variable is a placeholder to store any object or value. We discussed few naming conventions to make syntactically valid names. We also looked at few tips from Google R style guide for naming our variables. We also learned the use of assignment operator and assign function to assign some value to a variable. Then we discussed another important concept in our language, which is environment. So if we create a variable directly on console, then that variable will be created in global environment, but we can create custom environment also using new. env command, and we can create any variable in the custom environment. After discussing variables and environments, we talked about operators. We discussed various arithmetic operators such as addition, subtraction, multiplication, division and exponentiation. Then we learned more about some special numbers such as infinity, nan and na. We even realized that these special numbers help our language to continue to program execution or to terminate the program gracefully, even though we come across some adverse situations like overflow conditions. We even looked at various logical operators available in R language. Then in the last section of this module, we turned our attention towards vectorized operations and how they make R such a powerful language. We even looked at various flavors of vectorized operations where different operators or functions can work on one or more vectors and give results back without writing explicit loops. So in this course we looked at vectors that is one of the available data structures in R language. In the next module we will talk about various other kinds of data structures as well. Different data structures are used in different scenarios, and understanding data structures is a crucial part of learning R language. So see you in the next module.

R - Data Structures (Part 1)

Introduction

Hi, this is Abhishek Kumar, and welcome to the fourth module on R programming fundamentals. In the next couple of modules, we will be discussing various R-data structures. Data structures, are arguably the most important piece in your data analytics journey with R. In any data analysis project, you will be dealing with lots and lots of data, and data structures will define the way that data will be stored, and organized in the memory. So a solid grasp on data structures, will surely help you in your data analysis endeavors. We have divided various concepts involved in R-Data Structures in two parts, that will be covered in two modules. In this module, which is the first part, we'll provide you an overview of R-Data Structures, along with the details of several one dimensional R-Data Structures. While in the next module, which is the second part, we'll be focused towards higher-dimensional data structures. These data structure modules will not only help you to learn different data structures in R, but also guide you to use them in real-world scenarios. We'll be taking several demos also, to further understand various data structure concepts.

Outline

Here's the module outline. We will start this module with some fundamentals of data structures, then we will take various R-Data Structures in the detail. We'll be covering six mean R-Data Structures in this course. In the first part of the data structures module, we will be covering atomic vectors. Factors and Lists. In the second part, which is our next module, we'll be discussing Data frames, Matrixes and Arrays. In this module, we will not only cover fundamental concepts, but also will be taking a number of demos to substantiate our understanding on these topics. But before going into the details of different data structures, let's first discuss some fundamentals of the data structures in the next clip.

Data Structures in R

Data structure can be considered as a collection of data elements, which are grouped under one name. So, you can think of a structure as a container like this box. And selection of data structure mainly depends upon, answers to two very important questions. First question is, what kind of items do you want to put in this container? And the second question is, how to arrange different items in the container? Let's answer the first question, what kind of items to put in? Well, if you put similar type of items in the container, then you will have a homogeneous data structure. However, if you put dissimilar items in the container, then you will have a heterogeneous data

structure. Atomic vectors, matrixes and arrays, are homogeneous data structures, as these data structures contain items of similar type, while lists, and data frames are heterogeneous data structures, as these data structures can contain items if dissimilar type also. Now let's answer the second question, how to arrange different items? Let's first have a look at homogeneous data structures. So in the case of atomic vectors, items in the data structure are arranged in one dimension. In a matrix items are arranged in two dimensions, while an array can be n dimensional. For heterogeneous data structures, such as lists, the items are arranged in one dimension, while data frames are two-dimensional in nature. So far, we have looked at categorization of our data structures based on the types of items, and arrangement of items. Now let's have a brief look at different basic items you can put, in a data structure. Typically in a real world scenario, most of the data you will come across are either objects of some basic classes, or objects which can be built using those basic classes. These building blocks or basic classes, are also known as Atomic Classes. So let's have a brief look at some of the important Atomic Classes available. So we have characters, as one of the basic classes. Character objects are typically wrapped in single or double codes. Next, atomic class, is numeric class to work with numeric values. These numeric values are real or double values, and they can contain decimal part also. And our integer, is a separate atomic class. Integers do not contain any decimal power, other than integers, there is a logical atomic class, in which objects can be made of logical values, such as true or false. Another atomic class, is complex atomic class. If you have used complex numbers in your high school mathematics, then you know that a complex number has two parts. First is the real part, and second is the imaginary part. So if you have some items which can be represented as complex numbers, then you can use this complex atomic class. So in most of your data analysis projects, you will be working with objects made of these basic atomic classes only. Well, this was a short introduction to data structures in R. We now know that, what are different items which can be stored in a data structure, and what are different categories of data structures? Now let's take different data structures one by one, starting with atomic vectors in the next clip.

Atomic Vector

Atomic vectors are also commonly known as vectors. We have looked at vectors in the previous module also, where we learned about vectorized operations. As mentioned in the previous clip, atomic vectors are homogeneous data structures. Means in atomic vector, each element should be an object of same class only. So, a character vector, will have all elements of character type, while a numeric vector will have all elements of numeric type. Similarly, an integer vector will contain only objects of integer atomic class. A logical vector will contain only logical values, while

a complex vector will contain objects of complex atomic class only. And moreover, the atomic vector has one dimensional arrangement of items. So you can think that, all elements of the vector will be arranged one after another, in one-dimensional fashion. So far, we have discussed vectors, now let's see how we can construct vectors in R language. Here again, we will take the example of a fictitious classroom, with four students, Raj, Rahul, Priya, and Poonam,. Suppose, we want to create a vector to store names of all four students, to do this, we can create a vector student. names using combine function. In the combine function, we have passed all four names as arguments, so this vector is character vector, because every member of this vector is of character type. Well, we can use combined function to create a numeric vector also. Here we have created a vector student. weights, to store weights of all four students. Similarly, student. physics. marks is an integer vector, which contains marks of each student in physics subject. In our language, we use capital L as suffix to exclusively mark any numeric value without a decimal bar as an integer. So if you will not use L suffix with each element, then they will be treated as double values, and thus making this vector a numeric vector. Next, we have created a logical vector, student. physics. interest which contains four logical values, which represent whether that student has interest in physics or not. Also note that, for logical values, you can either use true or false in all capital letters, or you can use the short rotation of capital T or capital F. So this was a short introduction to Atomic Vectors. Now let's know more about Atomic Vectors, in a demo. Here we are, in our studio. Here on line number 1, we have created a character vector which contains names of all four students of fictitious classroom. So we have passed the names of each student to this combined function. Also, each character string, is wrapped in codes. Let's execute this line number one. So our vector has been created. Then you can use the variable name, to print the content of this newly created vector on the console, as we did in the previous module. However, I want to show you another very useful function which is a steer function. Steer function, can be used to display structure of any odd object. Here on the line number three, we have used STR function to see the structure of student. names object. Let's execute this line. As you can see here on the console, we get some information about this object student. name. So this is saying that, the object contains objects of character type, and it has four elements that index from one to four. This function will also show the contents of this object. So, STR function, is a very handy tool to explore the structure of any R object. Moving on to the next line, in R language there are several functions the labor to check the vector type. So on line number 4, we have used is. character function to test if the argument passed to this function is a character vector or not. Here we have passed the student. names vector, so this function should return true. Let's confirm it by executing this line number 4. And as expected, we got the output of true. Then on the line number 6, we have used combine function again, to create a numeric vector, student. weights,

and we have passed weights of all four students, as arguments to this combine function. Let's execute this line. So student. weights vector has been created. Now again, we can use STR function to see the structure of this object, as shown here on the line number 7. Let's execute this line. So as we can see, this object contains items of numeric type. Moving onto the next line, here we have is dark numeric function to test for numeric vector. Here we have passed the object student. weights, as the parameter. Let's execute this line, and as expected we got true in the result, as we had passed a numeric vector to this is dark numeric function. Again, I want to emphasize one point here, that in place of a vector, if you would have passed a scaled numeric value also, then also you would have got true in the result, because in our, a scale value is treated as a vector of lent one. Next on the line number 10, we have created an integer vector student. physics. marks. Here, we have explicitly used L suffix with each value to mark them integer type. Let's execute this line. So, our integer vector has been created, now again, we can use STR function to see the structure of this vector, as shown here on line number 11. Let's execute this line. So as you can see on the consul, we have an integer that's four elements. Next on the line number 12, we have used instruct integer function to test for integer vector. This function will return true, if the argument passed to dysfunction is an integer vector. Let's review this line. And as we expected, we got an output of true. Another important point is that, an integer vector can be considered as a numeric vector, but the reverse may not be true. Let's take an example. So if you use is. numeric function and passed the student. physics. marks vector to this function, as shown here on line number 15, then this function will return true. Because integral values, are basically numeric values without decimal part. However, if you use is. integer function is parse a numeric vector, then the output could be either true or false. Here on line number 16, we have parsed student. weights numeric vector, but elements of this numeric vector are in decimal, therefore, they cannot be termed as integers, and the output of this function will be false. Let's confirm it by executing both line number, 15 and 16. And, as expected, we have an output of true for the first statement, while an output of false for the second statement. Moving on to the logical vector, here on like number 18, we have created a logical vector student. physics. interest. And again, we have used combine function, and we have passed logical values. However, note that in R language, you can use the short notation of capital F or capital T, as well and true and false, in all caps to represent logical values. Lets execute this line, and I will watch the vector has been created. Now again, we can use STR function, on this logical vector to see it's structure. Let's execute this line number 19. So, we can see here that we have four logical values, indexed from 1 to 4. Moving on to the next line, there is is. logical function in our, to test for logical vector. Here are the line number 20, we have passed the newly created logical vector student. physics. interest to this is. logical function. Let's execute this line. And, as we had passed the logical vectors of the

function, we got a response of TRUE. On the similar line, we can create complex vector also. Here on line number 22, we have created a complex vector again, using the combine function. On line number 23, we have used STR function, to see the structure of the complex vector. And on the line number 24, we have used this dark complex function to test for the complex vector. Let's execute all these three lines at once. So first the complex vector was created. And then the str function displayed the structure of this complex vector. And finally, as dot complex function returned the output of true as we had passed the complex vector itself, the is dot complex function. So far we have used combined function to create atomic vectors. But you can also use vector command to create atomic vectors. Here from line number 27 to line number 31, we have created 5 vectors. In the vector command, we can pass the atomic class name, wrapped in codes, as an argument. So far creating the character vector, we use character in codes, for numeric vector we use numeric in codes. Similarly, you can use other atomic class names. Other than atomic class names, we can pass the length of the vector, here for example we have passed four as the value of the length parameter. So if you use this vector command, then you will get a vector with some default elements. So for the character vector, the default element value is an empty string while for numeric and integer vectors, the default element value is zero. For the logical vector the default element value is false and for a complex vector, the default element value is zero plus zero i. Lets executive all these five lines from 27 to 31 at once. So as you can see on the console, the character vector has four elements all set to empty string. The numeric and integer vectors have four elements with each element is 0 by default. Similarly, the logical vector contains four false values and the complex vector has all elements with a value of 0 plus 0i. So this was a brief introduction to atomic vectors. Now in the next clip, we will look at a few common operations on atomic vectors.

Common Operations on Atomic Vectors

In this clip, we'll take three Common Operations on Atomic Vectors. First is Arithmetic and logical operations. Second is Subsetting, and third is Coercion. Let's take a few examples in a demo to know more about these operations. Let's first look at some arithmetic and logical operations which you can perform on atomic vectors. In fact, we have covered such operations in the previous module, where we had discussed vectorised operations. Here on line number two and three, we have created two integer vectors, each containing marks obtained by four students in a classroom test on physics and chemistry subjects, respectively. Let's execute these two lines. So our integer vectors have created. Then on the line number four, we have used an additional operator to add both vectors to get total marks obtained by all four students. Let's execute this

line and as a result we get four different values each representing some of marks obtained in physics and chemistry subjects by each student. So here in this example, we have used the addition operator. But in the same manner, you can use other arithmetic operators also. Now let's look at one logical operation. On line number 5, we have used a logical operator to get all students who have got distinction marks of 75 in the physics subject. So here we have used greater than equal to logical operator. Let's execute this line, so here we have got four logical values with one false and three true values. These four values also form a logical vector of length four. So here I have used only a couple of operators in this demo, but you can play with different types of vectors as well as different types of arithmetic and logical operations. Now let's look at another very common operation that is subsetting. You can think subsetting as a process of extracting one or more elements from any data structure. And here as we are working on atomic vectors, so in this demo, our objective will be to extract elements from an atomic vector. To work on subsetting, let's first create an atomic vector. Here on line number eight, we are creating a character vector, student. names, using the combine function. And we have passed names of all four students. Let's execute this line number eight. So our character vector has been created, now suppose you want to extract the first element, or the first name from this student. names character vector. So for this, you can use a square brackets, and inside the square brackets, you can pass the index number. Remember, in art there is 1 based index, so the first element has an index of 1 and so on. So, if you'll pass 1 in the square brackets, then you can get the first element of the vector. Let's execute this line number 9. So as a result, we got the first element of the character vector, which is the first to remain Similarly to second element, you can use two inside the square brackets as shown here on the line number 10. Let's execute this line and we get the second name Rahul as a result. You can also extract multiple elements from the atomic vector. Here on the line number 11, we have used the colon operator. This statement means that, give me all elements from this vector which are indexed from one to three. So this statement will return three elements of the vector. Let's execute this line. And as a result, we got elements indexed at one, two and three. You can also use a logical vector to extract multiple elements from an atomic vector, as shown here on the line number 12. Here to create a logical vector, we have again used the combine function, and in the combine function we have paused four logical values. So you can read this whole statement as, give me first and third elements only, and don't give me second and fourth element. Let's execute this line, and as the result, we got two elements back which were positioned at the first and the third location. Subsetting using logical vector can be a very handy technique. Suppose we want to know, names of all those students who got distinction marks of 75 or higher in the physics subject. Here on the line number 13, inside the square brackets, we have used a logical operator over student. physics. marks integer vector. So this

term inside the square brackets will return a logical vector, where the elements will be true for all those who have got 75 or higher marks. And eventually this whole statement will give names of all students who got distinction marks in physics. Let's execute this line number 13. So we got Rahul, Priya, and Poonan as a result. We have scored marks greater than equal to 75. So as you can see, use of logical vectors in subsetting and atomic vector could be very handy and useful. So in this clip, we looked at few examples on subsetting. Now let's take few examples on coercions. Coercion means converting one type to another. Sometimes we also call this as typecasting or type conversion. Now, coercion can be both implicit or explicit. Take this example on the line number 17. Here we are creating a vector of student. weights using the combine function. And suppose, in the combine function along with three numeric values, you pass fourth value in codes. Let's execute this line number 17. So, our vector has been created. Now, let's again use the str function. As shown here on the line number 18, to see the structure of this newly created vector. Let's execute this line. So if you look closely here, the vector contains all elements of character type. It means, that the first three numeric values were implicitly caused to character data type. So whenever there is a mixing of data types, coercions So was a brief intro to implicit coercions, but coercions can be explicit also. And these coercions can either be sensible or not sensible, let's look at a few sensible coercions first. Here on the line number 22, we are trying to convert a logical vector to a numeric vector using as. numeric function. Typically in R, functions available to perform explicit coercion have a similar form. Means as, then dot, and then followed by the data type name. So here as an argument to this as dot numeric function, we have passed a logical vector. Let's execute this line. And as you can see here, we have results in numeric terms. So true logical values have been converted to 1, and false logical values have been converted to 0. Let's look at some more explicit coercions. Here on line number 23, we are using as. character function to convert an integer vector to a character vector. And R framework should not have any problem in doing so. Let's execute this line and as a result each integer value has been converted to a character string. Let's look at another coercion. Let's create the student. weights vector once again, but this time, we'll pass all numeric values to the combined function. So our numeric vector has been created, now on the line number 25, we are using as dot integer function to explicitly convert the numeric vector to an integer vector. Let's execute this line. So if you look at the output, you'll find that the decimal part from all numeric values have been removed. So you should always take extra precaution wherever coercions are happening. In fact sometimes coercions will not make any sense. Let's look at this line number 28. Here we are using as taught numeric function to convert a character vector, to a numeric vector. Let's execute this line. And, as a result, we are getting four NA values. We are also getting a warning message that NAs are introduced due to coercion. So make sure you are doing type conversions properly in your book.

So this was only about some basic atomic vectors. Now in the next clip, we will learn a special type of vector that is factor.

Factor

Factors are the special type of vectors which can be used to store nominal or categorical values. Categorical means the field can take a value from few categories only. Now let's take our classroom example once again. Now suppose you want to create a vector to represent gender of each student. Gender is a categorical variable because we have limited categories available. So as one solution to represent gender of all four students, you can create a character vector just like we did in a previous clip. But as gender is a nominal or categorical field, therefore we can also use factors in this case. And all you have to do is, to wrap the combine function output in a factor method. But what impact does it make, and what are its benefits? To know more, let's switch to our demo. Here we are in our studio, and our job is to store genders for all four students of the classroom. So here in the line number, one we have used the combined function to create a character vector, student. genders. And we have parsed gender of each student as characters in the combined function. Let's execute this line. So our character vector has been created. Let's print its contents using the variable name, as shown here on the line number two. Let's execute this line. And here, on the console, we can see all four character strings. So our job to store genders for all four students is completed. But for categorical or nominal values, this is not an efficient technique. Suppose later you have to filter your vector based on gender? Then you will have to perform string comparisons. String comparisons are always inefficient compared to individual comparisons. Also, you have to take care of the character cases as well. Because if by mistake there is a mismatch in the character cases, then comparison can lead to absurd results. So using integers to represent a nominal or categorical variable is a more efficient way. So alternatively, you can use an integer as shown here on the line number three. Here, for the female category, we have used an integer value of one, while for the male category, we have used an integer value of two. Let's execute this line. So, our integer vector has been created, now let's print its contents using the variable name as shown here on the line number four. Let's execute this line. So we have four integer values. Where one represents female category. And two represents male category. This technique is more efficient in terms of comparisons or filtering than using characters. But there is an obvious problem associated with this solution. These integers are not self describing. And you have to remember, what do you mean by a particular integer value. So here factors can come into the picture. Factors provide you the benefit of integers in terms of comprises, and they are also self describing. Let's create a factor ration here

on the line number five. Here we have simply the combined function with factor method, and inside the combined function, we have our character strings. Let's execute this line. So our factor is created. Let's print its contents by using the variable name as shown here on the line number six. So, if you look closely at the results. Here we have four values, and these four values are not wrapped in quotes. And moreover, we have another information of levels. So it means. We have two levels or two categories, these levels will also help you to ensure that, whether you have used consistent character cases or not while creating the factors. So this is how you create factors. Now let's look at a causal. Let's use as. numeric function to convert this newly created factor, into an integer vector as shown here on the line number seven, let's execute this line. Here on the console you can see that, we have four integers 2, 2, 1 and 1. So internally factors are integers only. However, labels help to make them self describing in nature. Now, you might be wondering why male level got the value of two, while female level got the value of one. This is because by default, levels are decided alphabetically. And because F comes before M, therefore we have this result. So, in this example, levels are created implicitly, but you can create explicit levels also. Explicit levels are used, if you want to order the levels on your own. You can also create additional levels, if you are using explicit levels. Here on line number eight we are creating a factor, a student. blood. groups. As the first argument we have used the combine function, and we have provided the blood groups for all four students. For the second argument, levels, we are again using a combine function. If you notice here that we have ordered the levels, as per our need and moreover, we have also included an extra level of B blood group in this case. So, additional levels are very useful, as it helps to provide all possible levels in a problem. Now let's review word this line. So, factor has been created, now let's use the STR function to see the structure of this newly created factor object, as shown here on the line number nine, lets execute this line. So here you can see that the factor is created with four labels in the order, which we had mentioned in the statement. Also, you can see the content of the factor, we had just created. Here the levels are displayed in integer values. So you should use factors for storing numeral or categorical variables, as factors arevmore efficient than character vectors, and they are self-describing than integer vectors. With the help of factors with explicit levels, not only you can order the levels, but you can also create additional levels, factors are widely used in data analysis projects. In fact some of the algorithms which you'll be using in future, they explicitly demand factors instead of character vectors. So this was a brief introduction to factors. Now we will look at another very important data structure, that is a list.

List

As discussed previously, list is a heterogeneous data structure. It means that you can put items of different classes or types, in a single list. And moreover, lists are one-dimensional in nature, so all elements of the list will be arrange in one dimension. So why do we need lists? Let's again take the example of our classroom, with four students. So far we have seen the usage of different types of vectors, to store different types of values. So we can use a Character vector to store names of all four students. We can us a Numeric vector to store weights. We can use a Factor to store genders for each student. And we can use Integer vectors, to store marks scored in physics and chemistry subjects. Now suppose you want to get all details off, the first student on the console. As we have learned in the vectors clip, we can use subsetting to extract one element from a vector. So we can use index 1 in a squared brackets to get the first student name. Similarly, we can use index 1 to get weight. Gender, marks in physics, and marks in chemistry. So as you can see, we have to write five lines of commands on the console to get details of one student. Well, here lists can come into the picture. With the help of a list, you can get the details of one student in a single command. So, let's see how we can create a list, and how it can be used. Before creating our list, let's first create our vectors. Here from line number one to five, we have statements to create five vectors. I have already described each statement in detail, in previous clips. So, I will not go into the details here, let's execute these five lines, so our five vectors have been created. Now let's create a list, here on line number eight and nine, we are creating a list variable named as student 1. This list will contain all details related to the first student. So to create a list in R, you can use list function. And in this function, we can pass all desired elements. And unlike vectors, list can contain items of different classes also. So the first element in this list, is the name of the first student, which is a character type. And to get the first name, we have used index 1, to extract the first element from the student. names vector. Similarly, the second element, which is a numeric type, is extracted from the student. weights vector. The third element is extracted from the factor, while fourth and fifth elements are in teacher type. So as you can see in this list, we have mixed data types. Moreover, as we have not specified names for each parameter, this list is an unnamed list. We will look at named list in just a moment. For the time being, let's execute this statement, so our list is created. Now let's use STR function, to look into the structure of this newly created list. Here we have passed the list variable name in the STR function, as shown here on the line number 10, let's execute this line. On the console window it is mentioned that it is a list of five elements. Data types of all elements are also mentioned. We can see contents of all elements also. So first element is character type, with the value of Raj. The second element is numeric type, with the value of 60. 5. Third element is a factor with two levels, male and female and value is two for the first student. Fourth and fifth elements are in teacher type, with values of 70 and 60 respectively. So as you can see we have mixed data types in this list,

now let's use the variable name to print this list content, as shown here on the line number 11, let's execute this line. Here in the console you can see the contents of this list. Indexed mentioned in double bracket, signifies element number. So contents of the first elements is Raj. Content of the second element is 60. 5 and so on. As you can see with each element, there is no associated name, but you can create named lists as well. Here from number 14 to 18, we have created a named list again using the list function, however this time, we have also given some name to each parameter, let's execute this statement. So our named list has been created, let's use STR function to see the structure of this list, let's execute this line number 20. Here on the console you can see after the dollar sign, we have names of each element along with its type and content. Name list could be really handy, to extract elements from a list. We will discuss subsetting of a list in the next clip. Well, so far we have used elements of atomic class only, as members in the list. But it is also possible to have a vector, as an element of the list, and for that vector a list can also be a member of another list. Here in the statements starting from line number 22, we have used a vector as a member of the list. We have used combined function to create marks vector, in which we have passed physics and chemistry marks as arguments, let's execute the statement. So, our list has been created, let's see the structure of this newly created list. Using STR function let's execute this line number 27, so now our list has four members and if you look at the fourth member, it is an integer vector with two members. Well this was a short introduction to the list. Now in the next clip, we will look at some common operations, that you can perform on lists.

Common Operation on Lists

In this clip, we will look at some of the common operations you can perform on lists. Primarily we will focus on subsetting in this clip, means extracting one or more elements from the list. But before doing that, let's first set up the scenario. Just like our previous clips, list first create few vectors. Here from line number one to five, we have created several vectors, let's execute these lines. So our vectors have been created. Now let's first create an unnamed list, so that we can perform subsetting. Here on line number nine and 10 we are creating a list student1, where we have included all details of first student, let's execute this statement. So our list has been created, now let's see how we can extract one or more elements from this list. On line number 12, we have used single brackets to extract an object. Remember or not, if you are using single brackets for subsetting, then you will get an object of the same type, from which you are subsetting. So if you subset a vector, using single brackets then you will get a vector. And if you subset a list using single brackets, then you will get a list. So on line number 12, we are using index of one inside single brackets and this should return a list, as we're extracting from a list. To confirm the return

type, we can use type of function available in R. On line number 13 we have used type of function and in this function, we have passed the first element extracted using single brackets. Let's execute both line number 12 and 13. So as you can see on the console, we have extracted the first element from the list. Now let's look at the type of the extracted element, it is a list. Now let's see what happens, when we use double brackets, to extract elements from the list. On the line number 14 we have used double brackets, and we have used the index 1 in double brackets. And on the line number 15 we are using type of function, to figure out the type of extracted object. So remember in R, if you are using double brackets for extracting element, then the return type will be the type of the element itself. Means if the first element of the list is character type, then using double brackets will return a character type object. Let's confirm the same by executing line number 14 and 15. So the first statement return the first element, and if you look at the result of the second statement then you will find that it is character type. And moreover, as we have a single value extracted here, so you can think the extracted object as a character vector of length 1. Now moving on to the next line in the script, if you want to extract multiple elements from the list, you can use colon or sequence operator also. Here on the line, number 16, we are extracting elements indexed from one to three. Let's execute this line. And as a result, we get a list of three items. I'm saying it as a list, because we have used single brackets for subsetting. So this is how you can perform subsetting on unnamed list. However, while extracting elements from the unnamed list, you have to remember the index, or the ordinal number, for each element. But if you use named list, then you do not have to worry about elements' order in the list. Let's first create a named list. Here we are using the statement, which has spanned from the line number 20 to 24, and we are creating a named list. We have given names to each element of the list. Let's execute this statement, so our named list has been created. Now we can use names to extract elements, instead of index number. Here on line number 26, we are extracting the student1 name. And we have passed the name, in quotes inside the double brackets. Let's execute this line, so we get the name of the first student. And as you can see using names to extract an element, is quite intuitive and convenient. You can also use the dollar sign, to extract element from a name on the list. Here on the line 27, we have used dollar followed by the element name, or label name. The benefit of using the dollar sign is that, you don't have to wrap the elements name in quotes. Let's execute this line, so we get the gender of the first student. Moving onto the next line, you can also use multiple names, to get multiple elements. Here on the line number 28, we had created a character vector, using the combined function, in which we had passed two names. So you want to get physics and chemistry marks, of student1 in one go. Let's execute this line, and on the console we can see marks obtained by student1, in both physics and chemistry. Well other than subsetting, you can also use the length function, to get the total number of elements in the list.

This function can also be very handy. Here on the line number 31, we have powers to the list variable student1, to the length function. Let's execute this statement, and we get an output of 5 on the console, because the list contains five elements. So in this clip we covered some basic operations, that you can perform on lists.

Summary

So we have reached to the end of this module. In this module we started off our journey by first understanding the data structure, and its requirements to arrange data in the memory. Then we learned that in our, there are two broad categories of data structures. In the first category there are homogeneous data structures, which can contain items of similar type only, while in the second category there are heterogeneous data structures, which can hold items of different types also. When we discussed various R data structures in detail, we started with atomic vectors, which are one dimensional in nature, and can contain similar type of items only. We looked at character vectors, nomatic vectors, integer vectors, logical vectors, and complex vectors. We learned to create these atomic vectors using combine function, as well as using the vector function. Then we looked at a special case of character vector, that is factor. Factors are useful if you have categorical or nominal fields. These factors are more efficient than character vectors, and self describing than integer vectors. After learning vectors and factors, we discussed another one dimensional, but heterogeneous data structure that is lists. Lists is very useful to combine different inter types in one place. We also learned to create a list, using list function. Well, throughout this module we also discussed various common operations, which can be performed on these data structures, to a number of demos. So as you can see that this was module was all about one dimensional data structures, but in a real world scenario, these one dimensional data structures may not be sufficient. Therefore in the next module, which is the second part of the data structures module, we'll be discussing some more R data structures, which can be helpful to you in dealing with various real world situations. So, see you in the next module.

R - Data Structures (Part 2)

Introduction

Hi, this is Abhishek Kumar, and welcome to the fifth module on R Programming Fundamentals. This module is the second part of Data Structures module. In the first part, we learned some

basics of R-Data Structures. We also looked at several one dimensional R-Data Structures in detail, such as vectors, factors and lists. So in this module, we will take our learning of data structures to the next level. We will take few higher dimensional data structures, which will be used in your various data analysis projects. By the end of this module, you will learn to create these data structures, and use them in our framework.

Outline

Here's the module outline. In the first part of data structures module we covered Atomic vectors, Factors and Lists. While in this module, we'll be taking Data frames, Matrix and Arrays. You will also learn to create and use these data structures in our, through a number of demos. So, let's take the first data structure in this module, that is, Data Frames in the next clip.

Data Frame

Data Frame is arguably the most popular data structure in R. It is a heterogeneous data structure. Means this data structure can also contain elements of different classes, just like the list. However, unlike list, data frame is 2-dimensional in nature. You can think data frames as your spreadsheets where you have different columns. Each column is a field and values will be stored in different data rows. Now let's see, why do we need data frames? Let's take our classroom example once again. So we first discussed, how with the help of different types of vectors, we can store different types of values. So we have to create Character vector to store names, Numeric vector to store weights, Factors to store gender and Integer vectors to store marks in physics and chemistry subjects. But then we found out that if you want to get all details of a single student then we have to extract elements from all vectors one by one. Then we discussed, how with the help of a list we can do it easy. And the list can help us to store different types of objects. And we created a list named as student1 and put all the required details in one place. But suppose we want to get details of the second student. Then again, we had to create another list for student2, for student3, another list, and for student four, one more list. It means that you have to create four lists to represent a classroom of four students. Creating lists might not be a tough task for a small classroom, but as the number of students will increase, it will become more difficult to create as many lists. Here, data frames can come into the picture. So now let's see, how with the help of a data frame, you can solve this problem. Before creating the data frame, let's first create our vectors which contain all details of four students. So from line number one to five, we are creating vectors using combine function. Let's execute these lines. So our vectors are created. Now to

create a data frame in R, you can use data. frame function as shown here on line number seven and eight. In this function, we can pass our atomic vectors directly to the data. frame function. However, one important point in data frame is that, each object passed into the data frame function should have an equal number of elements. And in this case, we do not have an issue, because each of our vector contains four elements. So a student. names vector contains four names, student. weight vector contains four weights. Student. genders contains four genders. Student. physics. marks vector contains four integer values. And finally, the student. chemistry. marks vector also contains four integer values. So this statement will create our data frame in one go. In this statement, we have explicitly created vector variables first, but this is not mandatory. You could have written four combined functions, directly in the data. frame function. But, there is a benefit of using vector variables, which we will see in a moment. Now, let's execute this statement. So our data frame has been created. Now let's look at one interesting thing about data frames. Let's try to get the type of this newly created data frame using typeof function. As shown here on the line number nine. Let's execute this line. You might be surprised to see the output here. Here we got the output of list. Because a data frame is basically a list where each element of the list is a vector of equal length. And in the students data frame we have used five vectors, each of length four, to create a list. Now let's use the variable name to see its contents as shown here on the line number ten. Let's execute this line. As you can see on the console, we have a two dimensional arrangement. And the vector that even names became our field or column names. This top line is also referred as the head. On the left hand side, R has automatically given each row an inclusion name. These are known as row names. We can explicitly mention row names also, and then we have our values in each data row. Each value in the data row is also called as a cell. Now let's see the structure of this data frame using the str function as shown here on the line number 11. We can pass this variable name as argument to this str function. Let's execute this line. Let's examine this output. On the left-hand side of the columns, we have column of filled names after the dollar sign. So by default, the variable names are used as filled names. These names are very handy and can be used in subsetting, which we will look in the next clip. Then on the right hand side of the columns, we can see types of each column along with column contents in a condensed form. So, student. weights is a numeric vector with four numeric values. Student. genders is a factor with two levels, male and female. And similarly, we have two integer vectors representing marks in physics and chemistry subjects. However, if you watch the first field, student. names, it became factor type, even though we had passed character vector. Well, this is a default behavior of data. frame function. So by default, character string values will be treated as factors even though student name is not a categorical or nominal variable. But there's a way around to remove this default behavior. You can set the Boolean parameters StringAsFactors to

FALSE as shown here. Let's execute this statement now. So our new data frame has been created. Now let's again use the str function to see the structure as shown here on the line number 17. Let's execute this line. Now you can see that the first field has become character vector again. Well, this was a short introduction to data frames in R. Now let's look at some common operations that you can perform on data frames.

Common Operation on Data Frames

In this clip, we will mainly focus on subsetting operations on data frames as they are most important piece while working on data frames. So let's create a data frame to work on. Again, we will take our classroom scenario. So just like our previous demos, let's first create different vectors to store name, weight, gender and marks, using the statement written from line number one, to line number five. Let's execute these lines at once. So our vectors have been created. Now let's create our data frame. Here on line number seven and eight, you're creating the data frame students, using the data. frame function. Moreover, we have set the value of stringsAsFactors to False so that the strings should not be considered as factors. Let's execute the statement so the data frame named as student, has been created. Now let's look at some sub-setting examples where you will learn to extract one or more elements from the data frame. Let's start with the statement at line number 11. Here we are using single brackets to extract the first element from the data frame. And if you recall from our previous clip learning, that if you use single bracket then it will return an object of the same type. Means, if students is of type list, then the extracted object will also be a list. So to confirm this, we can use typeof function as shown here on the line number 12. Let's execute both line number 11 and line number 12. So as a result of first statement execution, we got our first element, which is student names. We got all student names, as well as the heading, in our data frame. And moreover, you can see that the extracted object is a list because we had used single brackets while performing subsetting. Now let's use double brackets in place of single brackets, as shown here on the line number 13. And as we have learned earlier, this should return an object of its own type. Means, it should return a character vector. Again, to confirm the type, we are using the typeof function as shown here on the line number 14. Let's execute both lines together. So as a result of the first statement, we got our names. And if you look at output of the second statement, we can see we have an output of a character vector. So in this example we used indexed numbers to extract elements but you can also use column names to do so. On line number 15, we have used column name of student. names instead of the indexed number one. Let's get the result of this statement along with a type of extracted output using type of function on line number 16. And as expected, we got all the student names as a result

from the execution from the first statement and we got an output of the character vector from the second statement. Alternatively, we can use the dollar sign as shown here on the line number 17. The benefit with dollar sign technique is that, you do not have to grab the column name in codes. Again, you will get the character vector if you will execute the statement. Let's confirm it by again using the typeof function as shown here on the line number 18. Let's execute both line number 17 and 18. So again, we got to see more but with four names and the type of destructed object is a character vector. So far we have discussed techniques to extract one column of the data frame only. Now let's see how we can extract multiple columns. Suppose you want to get a slice of the data frame students, with columns from one to three. So to do that, you can use colon or sequence operator as shown here on the line number 19. Inside single brackets, we have written 1:3. And we're using single brackets so the output will be a list which in turn will be a slice of the data frame. Let's execute the statement. And we get the output with three columns. So we have simply cut our data frame vertically and have extracted first three columns. Here in this case, we have used index numbers. But we can use column names as well as shown here on the line number 20. So instead of single brackets, we can use combine function to create a character vector in which we have used column names. So this statement means that, give me the slice of the data frame students with two columns, student. physics. marks and student. chemistry. marks. Now let's execute this line, and as a result, we got the desired output with two columns. So far, we are extracting whole columns, but we can extract individual cells also. But before that, let's split the data frame once again, using the variable name as shown here on the line number 21. So here is our data frame. Now suppose you want to extract the weight of the first student Raj. So for that here on line number 22, we have used one comma two inside single brackets. The first number represent row number, while the second value after comma represents the column number. So here, we are trying to extract the cell value located at the first row and second column, let's execute the statement. And we get the double value of 60. 5 as a result. Here we are extracting a single cell only, but you can extract multiple cells also. Here on line number 23, we are extracting multiple cells. For row number five, we are using the sequence operator. Here, we are trying to extract sales from row number one to three. Similarly for the column number also, we are again using a sequence operator. And here we are trying to extract sales from column number one to two, let's execute the statement, so here is the result with three rows and two columns. So you can use colon or sequence operators to select consecutive rows of columns. But you can also select individual rows or column numbers. Here on line number 24, we are trying to extract row number one and two, and column number one and three. Let's execute the statement, and here's the result with two rows and two columns. Now let's look at a few more subsetting examples. Here, on line number 25, you have specified the column number 1, but you have not

specified row number. This statement means, get all values into column number 1. Then, on line number 26, you have specified row number as 1, but we have not specified the column number. This statement means, get all values in a row number 1. Let's execute both lines. So as a result of the first statement, we get all rows in the first column means we got all four names here. While as a result of the second statement, we get all column values for the first row. Means, we got all information for the first student. You can also use logical vector to pick required rows of columns selectively. Here on line number 27, we have used a logical vector with four logical values before the comma symbol. This statement means that, give me the first and third row, and don't give me second and fourth row. We have not specified any column number, so it means that select all columns. Let's execute this line. So as a result, we got only the first and the fourth row with all five columns. This technique of using logical vector can be very helpful in filtering the data frame based on certain conditions. Here on line number 28, we are trying to get details of all male students. To do this, we have used a logical operator equals two. And we have compared the gender column name with male value. This is a vectorized operation and this will result into a logical vector. So this statement will give all male student details only. Let's confirm the result by executing this line. And on the console, we can see we have details of only male students. Similarly, on line number 29, we are trying to get details for all those students who have got distinction marks of 75 or more in Physics subject. Let's execute this line also. So here we have got three rows of results for which physics marks are greater than or equal to 75. Well, in this clip we went through various techniques for extracting one or more elements from the data frame. These techniques will be very handy while working with data frames in future. Well, in most of the data analysis projects, we will be using vectors, lists, factors, and data frames. However, I want to give you a brief introduction to matrixes and arrays also. So let's take matrixes first in the next clip.

Matrix

Matrixes are similar to data frames as matrix also have two dimensional arrangement, just like a spreadsheet. However, unlike data frames, matrixes are homogeneous data structures. Means, a matrix can contain items of similar type only. Typically, we use matrixes to store and process numeric data. Let's take our classroom example once again. Suppose we only have two student marks of these four students in Physics and Chemistry tests. Then we can use a matrix also to store these values, as all these values are in Decher types. Now lets take a quick demo to see how we can construct a matrix in aug. Before constructing a matrix, let's first create our two vectors. Here on line number one and two, we are creating two integer vectors. These vectors contain

marks obtained by four students in physics and chemistry subjects, respectively. Let's execute these two lines at once. So our integer vectors are created. Now let's create our matrix. Well, matrixes can be created using different techniques. Let's take few of them here. Here on line number 3, we're using the rbind function to create the matrix student. marks. Rbind in basically row binding. In the rbind function, we have parsed our two integer vectors. Let's execute this line, so this line has been executed. Now let's see the content of the newly created matrix using the variable name action here on the line number four, let's see this resulting matrix. As you can see, both physics and chemistry marks are simply stacked over each of the row values. So physics marks became the first row, while the chemistry marks became the second row. These variable names on the left are known as row names. Here we can see a few autogenerated column names. If you look at the format of the column name, it is comma, followed by index number. And if you remember from the previous clip, the value after the comma in a square bracket represents the column number. So using the rbind, you can perform row wise binding. But you can perform column wise binding also. Here on line number five, we are doing a columnwise binding using the cbind function. Again, we have passed both integer vectors to this cbind function. Let's execute this line. So our matrix is created. Now let's see its content, again using the matrix variable name as shown here on the line number 6. So if you see the output. Here you can see values are stacked column wise. So physics marks became the first column while the chemistry marks became the second column. The variable names are column names or head, while row names are auto-generated. And here index number are before the comma symbols. As you know that the first number in the square brackets represent a row number. You can also have your own custom row names as well. Let's add custom row names on line number seven, on the left hand side of the assignment operator, we are setting the row names by parsing the matrix in row names function as an argument. And on the right hand side, we have our row names where we are using combine function to create a character vector. In this character vector, we have used names of all four students. Let's execute this line. So row names have been set. Now let's see the content of this matrix using the variable name. Let's execute this line number 8. As you can see, we have our custom row names on the left hand side. Now let's see the structure of our matrix using str function as shown here on the line number 9. Let's execute this line. Let's examine the output. So this matrix contains all integer values, rows are indexed from 1 to 4, while columns are indexed from 1 to 2. You can see its contents also. Another important property of matrix is dimnames. Dimnames or dimension names is a list which contains row names and column names. Now let's see another way of constructing a matrix. On line number 10, we are using the matrix function to create the matrix student. marks. As the first parameter to this matrix function, we are using the combined function to create an integer vector. In this combined function, we have parsed our

eight integer marks, scoring two subjects, physics and chemistry, by four students. Then, we have mentioned ncol, or number of column, to be two. And finally, the nrow or number of rows has been set to four. So this statement will automatically arrange these eight values in such a way that we get two columns and four rows. Let's create this matrix and then see its content using the variable name as shown here on the line number 11. Let's execute both these lines at once. So let's examine this resulting matrix. As you can see, both row names and column names are auto generated, and values are arranged to create a four by two matrix. Means, a matrix with four columns and two rows. Remember in R, if you use matrix function to store a collection of elements, then by default, the elements will be arranged column-wise. So first of all, the first column will be filled, then the second column will be filled and so on. But you can arrange items in a matrix row-wise also. Here on line number 12 and 13, we have set one extra parameter, that is by row. We have set the value of by row property to true. And when you set this by row property to true, then values will be arranged row-wise. Also in this statement, we have set the number of columns to be four and number of rows to be two. So, let's execute these two lines. So our matrix has been created. Now, let's see its content using the variable name as shown here on the line number 14. Let's execute this line. So, as you can see in the output, the values are arranged row-wise, and we have two rows and four columns. Well, this was a brief introduction to matrixes. Now in the next clip, we will look at a few common operations you can perform on matrixes.

Common Operation on Matrices

In this clip, we will look at some common operations on matrixes. Let's first set up the scenario. So let's first create a matrix, we will take the same matrix which we have constructed in the previous demo. So on line number 1 and 2, we are creating two integer vectors to store marks obtained by four students of a classroom in physics and chemistry subjects. Then on line number three, we are constructing a matrix, student. marks, using the cbind function. We have passed both integer vectors to this cbind function. Then on line number 4, we are also setting row names for each row in the matrix. To set row names, we are using a character vector in which we have passed names of four students. So now let's execute line number 1, 2, 3, and 4 in one go. So our matrix is created. Now let's print its content using the matrix variable name as shown here on the line number 5. So as you can see, we have our matrix ready with four rows and two columns. Now let's talk about process of subsetting or extracting one or more elements from the matrix. Techniques of extracting elements from a matrix are almost similar to that of data frames, which we had discussed in the data frame clips. However, let's take a few examples. In Matrix, mostly you will be using single brackets with comma to extract multiple cells. So, before the comma you

can specify the rule numbers, and after the comma you can specify column numbers. On line number 8, we have not specified any row number or column number. So, by default this will return the whole matrix. Let's execute this line. And as expected, we got the whole matrix with four rows and two columns. Next, on the line number 9, we have used 2, 2 in the single brackets, so this will return cell value at row number two and column number two. Let's execute this line. And we get a single value of 70, which is marks of 10 by Rahul in chemistry subject. Then on line number 10, we have not specified the column part, and we have specified two in the row part, so this will return all values in the second row. Let's execute this line. And as expected, we got the result of 75 and 70. Similarly, on the line number 11, we have mentioned only the column number. So this statement will return all rows of the second column. Let's execute this line, and as a result, we got chemistry marks for all four students. We can also use colon or sequence operator to select sequence of rows or columns. As on line number 12 in the row part, we have used one colon three. We have not mentioned anything in the column part, so this statement will return rows indexed from one to three. And all columns will be included in the result. Let's execute this line, and as expected, we get three rows with both columns in the result. We can also use integer vector to choose rows or columns selectively. Here on line number 13, we are extracting only first and third row with all columns. Let's execute this line. And as a result, we get only the first and the third row of the matrix. Instead of an integer vector, you can also use a logical vector as shown here on the line number 14. Here we are trying to extract only the first and the fourth row because we have true value in the first and the fourth additions only. Moreover, we want to include all columns in the result. Now let's execute this line. And in the result we get only the first and the fourth row with both columns. So in this clip, we looked at few examples of subsetting, but you can try different things of your own. Now let's look at few more handy functions that you can use with matrixes. These functions can be used to get row-wise of columnwise summaries. But before doing that, let's print the students. marks matrix once again by executing the slide number 17. So, here we have our matrix. Now suppose we want to get total marks scored by each student. For this, we can use rowSums function as shown here on the line number 18. This will add all values in a row. So this is a row-wise sum. Let's execute this statement. And we get sum of marks obtained in physics and chemistry for each student. Similarly, we can get column-wise sum also using colsum function as shown here on the line number 19. Let's execute this line. Here we have sum of all marks obtained in physics and chemistry subjects. Well, this doesn't make much sense. It would be better if we can get average marks scored in these subjects. So for that, you can use colmeans function as shown here on the line number 20. Let's execute this line. And as a result, we get columnwise mean. So average marks scored in physics subject is 77. 5, while average marks score in chemistry subject is 71. 25. Well, in this clip, I have shown you some

common operations on mattresses which you may encounter in data analysis projects. However, you can also perform various other matrix related operations, such as various arithmetic operations, inverse and transpose. You can also look into the R documentation if you have a specific requirement. So this was a brief introduction to matrixes. Now in the next clip, we will take one final data structure, that is array.

Array

Arrays are another type of data structure supported in R. You will not often encounter arrays in your data analysis projects. However, I want to give you just a glimpse of arrays here. Well, arrays are also homogeneous data structures just like vectors and matrixes. Means and array can only contain similar type of items. However, while vectors are one dimensional and matrixes are two dimensional, arrays on the other hand can be n-dimensional also. Let's extend the scenario which we looped in the matrix clip, where we had used a matrix to store marks obtained by four students of a classroom in two subjects, physics and chemistry. So this matrix can be thought of as a one-two dimensional sheet for one classroom. Well, there is no issues as such if you have only one classroom. Now, suppose you have multiple classrooms, as shown here. Then you will require more such sheets, means more dimensions will be required, so arrays can be used in such cases. Arrays can be considered as multiple sheets, or multiple matrixes stacked over each other. Let's take a simple demo to create an array in such scenario. You can construct array's in multiple ways, however, in this clip we will first create multiple matrixes and then we will them to create an array. Here from line number 1 to line number 4, we are creating our first matrix, using C bind function. I will not go into the details of each line here, as we have covered these lines in the matrix clip. So our first matrix, is class one dot student dot marks. And it contains marks scored by four students of first classroom in two subjects, physics and chemistry. Now let's execute these four lines. So our first matrix is created, let's print it's content using the variable name as shown here on the line number 5. So here is our first matrix with four rows and two columns. Similarly, we can create another matrix, which contains marks obtained by four students of the second class room, in two subjects. Let's create and print the second matrix by executing the lines from seven to 11. So this is our second matrix with four rows and two columns. Then on line number 13, we are constructing our array, student. marks, using the array function. As the first parameter to this array function, we are combining both matrixes using combine function, then we have set our second parameter, dim, or dimension. We have set the dim perimeter to an integer vector containing three individual values. First number represents number of rules in one sheet, second number represents number of columns in one sheet, and the third digit number represents

number of sheets itself. So, in this example we have four rules and two columns per sheet, and there are two such seeds as we have two classrooms. Now let's execute this line. So our array has been created. Now let's see its contents by using the array variable name as shown here on the line number 14. Let's execute this line. As you can see on the console, the first sheet contains marks of the first classroom students, while the second sheet contains marks of the second classroom students. So this is how you can construct arrays. Now let's quickly look at few examples to extract one or more elements from an array. So in an array you have three numbers to play with. First is the row number. Second is the column number. And the third is the sheet number. So a statement here on the line number 15 will return value at row number two, and column number two of the second sheet. Let's execute this statement. So as a result we get the value of 71. So this is how you can extract any element from an array. In arrays, you can also use other subsetting techniques, which we have discussed in the previous clips. Such as, like here on the line number 16, we are extracting rows from 1 to 3 and column number 2 for all sheets. Let's execute this line number 16, and we get the desired result. So you can play with arrays in different fashions. Well, this was a brief introduction to arrays, and some common operations on them.

Summary

So we have reached to the end of this module. In the previous module, which was the first part of data structures, we learned the difference between homogeneous and heterogeneous data structures. Then we discussed several data structures in detail, starting with the homogeneous data structures such as atomic vectors and factors. And then we learned the use of heterogeneous data structure lists. While in this module, which is the second part of the data structures, we started with data frames. Data frames are arguably the most used data structure in data analysis projects. Data frames are two-dimensional in nature, and can contain items of different types. You also learn the use of data. frame function, to create data frames. Then, we discuss few sub-setting techniques to extract one or more element from a data frame,. After learning the data frames we discussed a two dimensional homogeneous to the structure that is matrix. Matrixes can contain items of similar type only, and are particularly used to process neumatic data. We learn different techniques to create a matrix, such as using C-bind function or R-bind function. Matrixes can also be created using matrix function, after matrixes, we learn another homogeneous to the structure that is array. Arrays can be n dimensional also, so you can think arrays as multiple sheets where each sheet represent a two-dimensional matrix. So this was all about data structures. Well, so far in this course have used several invert functions to perform various kinds of tasks such as getting help or creating data structures, or getting object

structures. But in the real world scenario, you will be required to write custom functions also. So in the next module, we'll be working with functions, which is yet another pillar block in any programming language. In the next module, you will learn the requirement of functions, in real-world projects and how to write them in our framework. You will also learn about various nitty-gritties involved with functions. So, see you in the next module.

R - Functions

Introduction

Hi. This is Abhishek Kumar, and welcome to the sixth module on R Programming Fundamentals. In this module, we will talk about functions. R language has thousands of inbuilt functions, and we have already used several functions to perform various kind of tasks, so far in this course. Well, this module will take your understanding on functions, to the next level. We will look into various nitty gritties of functions in context of our language. By the end of this module, you will learn to create your own functions. You will also learn about various components of a function, and how to use them, in real world scenarios.

Outline

Here's the outline of this module. In the first section we will first have an overview of functions, and why functions are required at the first place. We will talk about different components of a function, and we will also look at couple of guidelines for consistent function naming. Then in the second section we will talk about different concepts, related to arguments. We will look at different ways of argument matching if you have multiple arguments in a function. We will also discuss a few spatial arguments such as, arguments with default values and initial arguments using ellipses. Finally, in the last part of this module, we will discuss functions as first class R objects, and how we can look into them, assign them, and pass them as arguments. We will also learn Anonymous Functions. So let's start with functional review, in the next clip.

Functions Overview

So far in this course, we have used several inbuilt R functions on several occasions. Every function we have used so far work in the same way. We passed some inputs to those functions, and the

function performed some tasks. And most of the times, the function returns some values also. But other than using inbuilt functions, you can also create your own custom function as well. So why do we need to create functions? Let's take our classroom scenario once again, which we had used in the previous modules also. So we have four students, Raj, Rahul, Priya, and Poonam, in an imaginary classroom. Now suppose these four students took a classroom written quiz on physics, and scored these marks. Then they all have to give viva test and they scored these marks respectively. Now we want to get total marks as scored by each student in the subject. Here we are simply adding both quiz and viva marks to get total marks, and then assigning the total marks to our variable. High while there could be other things also involved, while calculating total marks. Probably there could be some normalization stuffs and all. However, here we want to keep it pretty simple. So this line here is doing the main part of calculating total marks, so once we are done with our main logic of finding total marks, we can get the final marks using the variable for total marks. This whole process might look okay, but it has few problems. Now suppose, we want to use the same logic for calculating the total marks, in another subject. Suppose students of the same classroom have taken tests in chemistry subject also, and this code, these marks, increase. And these marks in viva. Now, again we want to get total marks as scored by each student in chemistry subject also. Well, again we have to write the logic for calculating total marks, like this. And after that we can get total marks. So again you can see here, we again have to write the logic for calculating total marks. So what will happen if logic of getting the total marks would have been more complex or more lengthy in nature. Then, we need to write the logic again, and again. Here functions can come into the picture. Let's see how we can use a function to simplify this whole process, and make it more reusable.

Demo: Functions Overview

Here, we are creating a function, GetTotalMarks. After the function keyword, we have some values in the parenthesis. These values are called as, arguments. And then we have few lines of code, between two curly braces. These lines are called body of the function. On the first line we have added the quiz. marks and viva. marks, and assigned the summation result to another variable total. marks. As you can see, these variables used on the right hand side of the assignment operator, R-function arguments. After finding the total marks, we have simply used the variable name. Remember in our, the last line represents the return value of the function. If you have in any other programming language, then you might have used return keyboard to return some value from the function. However, in ours, it is not mandatory. We can simply put the return value on the last line of the function, and the function will return it. So the lines here from line number one

to five, will create a function GetTotalMarks, which takes two values as arguments and return the sum of those two values. Lets execute these lines, so our function has been created, now let's see how it can be helpful to us. Here on line number 7 and 8, we have two integer vectors representing marks of four students in physics quiz and viva exams. Let's execute these two lines, so integer vectors are created. Next on line number 9 and 10 we are using our function GetTotalMarks. In this function we have passed two integer vectors as arguments. First integer vector represent physics quiz marks, while the second integer vector represents physics viva marks. We are also assigning the output of this function to the variable on the left-hand side of the assignment operator. So, when this line will be executed, the first value of this function will be passed as the first argument of GetTotalMarks function, while the second value of this function will be passed as second argument of GetTotalMarks function. So in this line we are calling the function and passing the values, to the arguments. This process is also known as function and invocation, or function call. Now let's execute this line. So this line has been executed, and the output of the function has been assigned to the variable student. physics. total. marks. Now let's see the content of this variable, using the variable name as shown here on the line number 11. Let's execute this line, and on the console we have our result. So now we know how to create a function and how to use a function. Now let's choose the same function to GetTotalMarks in Chemistry subject. On line number 13, we have an integer vector containing quiz. marks, often by full students in the Chemistry subject. And on line number 14, the integer vector represents chemistry viva. marks. On line number 15 and 16 we are again using the same function GetTotalMarks, but this time we have passed Chemistry quiz and viva marks in the function. So, as you can see here, we are reusing the implementation of getting total marks through the GetTotalMarks function. Let's execute these four lines. So our calculation is done. Now let's see the result, using the variable name. As shown here on the line number 17. Let's execute this line. And on the console we've got total marks by each student in Chemistry subject as well. So you can see the benefit of using functions, although this example our function is pretty simple, but it can be very complex also. In fact, functions also help to provide an abstraction layer. So far in this course we have used several in-built R functions. But, we do not have to worry about, internal implementations in those functions. We simply pass the values to the function and function will do some processing and can return some values. So the function caller need not to know the function internal implementation. And thus, functions are also helpful in providing an abstraction. So in this clip, we'll learn how with the help of a function, we can achieve reusability and abstraction. Now let's discuss different components of a function.

Function Components

A function has different components. Here is the function from the previous clip. Let's look at different components of this function. The variable on the left hand side of the assignment operator is the function name. This name will become the identity of this function. Later you can use this function name, to invoke or call the function. On the right inside of the assignment operator, we have function keyword. After the function keyword we have few variables in parenthesis. These are known as arguments to the function. The lines in the curly braces, represent the body of a function. So inside the function body, we write our function logic. So as you can see in this function there is no explicit return keyword to return some value, from this function. Because in R by default the last line of the function is treated as a return statement. Means, the last line of the function can act as implicit return value. However R also supports explicit return values, using return key word, which we will see later in this module. Well, now you know how to create a function, and what are its components. But before moving ahead let's discuss a couple of naming guidelines, for your function names in the next clip.

Function Naming Guidelines

It's always better to follow some guidelines or conventions for variable names in your project, and the same thing applies to the function names also. You can have your own guidelines also, for your project, however, you can also follow Google R style guide. As for Google R style guide, function names, should have initial capital letters, means, the function name should be in Pascal case. And there should not be any dots, or underscore in the function name. Look at these four function names. Out of these four names, the first one can be termed good,. Because, here we have used Pascal casing, and there are no dots or underscores in this name. However, second function name is not appropriate as it does not have appropriate casing for the first letter. Third and fourth function names are not appropriate as per the Google R style guide, as these names contain dot and underscore characters in-between. So you can follow these naming guidelines for uniform function names in your projects. Now from the next clip, we will talk about different concepts related to arguments. And we will look at argument matching if you have multiple arguments in a function.

Argument Matching

Argument matching or argument mapping, comes into the picture when you have multiple arguments in your function. Suppose you have a function MyFunction, with more than one

argument. And then you will invoke or call this function. Then you will pass required values for the arguments. So argument matching can be considered as mapping between, values passed from the function invocation, to arguments of the function definition. In R, this argument matching can be done in several ways. Let's look at couple of ways. First argument matching technique is matching by position. Let's take an example. Here we have a function, GetTotalMarks, with two arguments, quiz. marks and viva. marks. So then we will invoke, or call this function, when the value is passed in this function call, will be matched as per the position. So the first value will be passed as the first argument, while the second value will be passed as second argument. So in this example arguments are matched by position. However, this is not the only technique. You can also match arguments by name. Let's take an example to understand this. Here we have the same function with two arguments. However in this function call, we have also included, argument names. So this line here means, that pass the first integer vector as quiz. marks, and pass the second integer vector, as viva. marks. The benefit of using argument names in the function call is that, you can reorder your values in the function call. Let's look at this function call. Here we have reordered the values in the function call, but with the help of argument names in the function call, the R framework can successfully map these values to the function arguments, in the function definition. So this was about argument matching. Now in the next clip, we will discuss arguments, with default values.

Arguments With Default Values

An argument with some default value, is also known as the default argument. This is an argument which is not required to specify in the function call. Let's extend our classroom scenario. Suppose other than quiz. marks and viva. marks, there is a third component of marks as well, that is assignment marks. And suppose the teacher gives default marks of 5, to all students who have submitted their assignments on time. But if assignments are not submitted on time, then teacher can decide to give different marks to each student as well. So let's see how we can incorporate assignment marks in this GetTotalMarks function, with the help of default argument. Here we are in R studio. Here we have extended our GetTotalMarks function. If you look at this function, we have another argument, assignment. marks, and we have assigned a value of five to this argument in the function definition itself. This is how you declare a default argument in a function. So here we are setting the default value to this argument, assignment. marks, and inside the function body, we are adding the assignment marks along with quiz and viva marks. So this is our function implementation, now let's create this function by executing the lines from one to four. So our function has been created. Next, on the line number six, we are calling the function. But if you

look closely at this function call, we have only passed values of two arguments. But then also this function call will work because by default the value of 5 will be used for assignment. marks argument. Now let's execute this line and see its result. And as you can see here, we have total marks for each student. So in this case, the default value of 5 has been added to quiz and viva marks. But you can override the default value also. And can provide other values as well. Suppose the teacher later decided to give different assignment marks to each student. Then that is also possible. Here on line number 7 and 8, we are passing the value of assignment. marks argument explicitly in the function call. And we have passed an integer vector with four values here. So in this case, the default value of 5 will not be used. Rather, this integer vector here in the function call will be used. Let's execute the statement. And we get the final sum on the console. So, in this manner, the default arguments can be used in your R programs. Default arguments are very common in R. In fact in many of the inbuilt R functions, you'll find default arguments. Well, so far, we have seen cases where we have fixed number of arguments in the function definition, but let's see how we can deal with the scenarios where we are not sure about number of arguments.

Additional Arguments Using Ellipsis

Ellipsis is a very neat technique to represent additional arguments in R. It is represented by three dots. Ellipsis is also one function argument, however, you can think this as a special argument which means, anything else. So suppose you have a function with few known arguments, but in the function you want to use some more arguments, but you are not sure what those arguments could be. Users of this function can put different arguments to this function call. In those scenarios, ellipsis can be used as an argument to the function. Let's take a couple of scenarios where it can be very helpful. Let's take our classroom example once again and extend the GetTotalMarks function. Suppose the teacher has provisioned some fixed marks to each student for other things, such as creativity or attendance, other than quiz, viva, and assignments. But we are not sure about those additional arguments right away. So in such scenarios, we can use ellipsis or triple dots as an argument. Remember, in R, ellipsis should be the last argument in the function. So on line number two, we are using the ellipsis argument. Here we are passing the ellipsis argument to sum function. So this line means that first add quiz, viva, and assignment marks for each student. And then add the sum of all extra arguments to each student total marks. Now let's see its usage. On line number eight and nine, we have passed one more argument to this function call, that is, creativity. marks. And we have set its value to 2. So this additional argument will be mapped to the ellipsis argument. Then, on line number 10, 11, and 12, we have added one more argument other than creativity marks. Suppose the teacher decided to give

three marks to each student for attendance. So now we have two additional arguments mapped to ellipsis argument. However, one important point to note here is that these additional arguments are mapped through ellipsis argument not because they are at the end of the function call. Rather, these additional arguments cannot be mapped either using position or using name. It means you can again reorder these arguments and can still make use of ellipsis feature. So we can see that how ellipsis helped us to use the same function definition even if we have additional arguments. Now let's execute these lines here from line number 1 to line number 12 to create the function and then see the results of two function calls. So first our function is created. Then in the first function call, only the creativity marks are added to the total marks, while in the second function call, both creativity and attendance marks are added to the total marks. So in this manner, you can pass additional arguments to the function without changing the function definition. Typically, ellipsis are used to pass additional arguments to some external function. Here in this example, we have passed additional arguments to sum function. But you can work on these additional arguments also. So if we want to see or use the additional arguments explicitly, then you can wrap ellipsis in list as shown here on line number 3. Let's uncomment this line number 3. And let's also print these additional arguments on the console using print function as shown here on the line number 4. Let's uncomment this line also. Now let's create our function once again, and then execute both function calls. By executing lines from line number one to line number 12. So, first our function is recreated. Next, look at the result of this first function call. Here you can see in the console, we have one member in the extra arguments list. We have the dollar sign here before the creativity. marks. So you can use this in your program directly if you want to. Similarly, for the second function call, we have two members in the extra arguments list. First member is the creativity. marks and the second member is attendence. marks. So in this manner, you can use ellipsis to pass additional arguments and to work on them.

Lazy Evaluation

R supports lazy evaluation also. In lazy evaluation, evaluation of an expression is deferred until it is used for the first time. Lazy evaluation is very common in functional programming languages. Now let's see an example to understand lazy evaluation in the the context of function arguments. Let's take a hypothetical situation. Where, in the function GetTotalMarks, we have three arguments. First two arguments are quiz and viva marks, however, the third argument, extra. marks, is a default argument. But we have set the default value of this extra. marks argument to another variable, which is average. viva. marks. So here we want to add the average of viva marks to all students as extra marks. And we have set the value of average. viva. marks in the function

body. And we have used mean function to get average viva marks. After that, we have added this extra marks to each student total of quiz and viva. And finally, we are returning the total marks. Let's create this function by executing lines from line number one to five. So our function is created. Then on line number 7, we are making a function call, and we have supplied values of quiz and viva marks only. We have not passed any value for the default argument extra. marks. Now let's walk through the function execution steps. So these two values will be passed as arguments to the function. But for the default argument extra. marks, average. viva. marks will not be available at this moment. But then also, R will not raise any error at this point. This argument extra. marks will be evaluated only when it will be first used. This behavior is known as lazy evaluation. Now let's move further to see its benefit. Let's move to the line number two. And here, the variable average. viva. marks will be set to the average, or mean value, of the viva marks. Now let's move to line number three, and here, where R will discover that we have a new variable, extra. marks, which is coming to through the argument. Then it will go back to the line number one to find about extra. marks. And as extra. marks is equals to average. viva. marks, so it will search for average. viva. marks. And at this moment, we already have got average. viva. marks due to execution of the line number two. So finally, total marks will be evaluated without any issue. If lazy evaluation feature would not have been there in R programming language, then the language would have raised the error right away, at first line itself. Now let's execute this function call at line number seven to see the final result. And here we have got the final total marks. Without any error message. So as you can see, lazy evaluation can be very helpful to you in many scenarios. Well, so far in this module, we have written only those functions which are returning only single value. Now in the next clip we will learn to return multiple values from a function.

Multiple Return Values

So far in this module, we have looked at the functions which return only a single value. And we learned that there is no need of explicit return statement. And by default, the last line of the function will be treated as the return value. However, if you have to return multiple values from a function, a neat way to do this is to use a list with the return statement. Let's take an example to see this technique. Here, we are creating a function GetMarksSummary. The job of this function is to take quiz and viva marks and return two things. First is the total marks obtained by each student and second is the average marks scored in the subject. And as you can see, we have two arguments in this function, quiz and viva marks. Then on line number two, we are calculating total marks by adding quiz and viva marks. And on line number three, we are using the mean function to get an average of total marks. So these are the two values which we want to return from this

function. So to do this, we have used the return statement as shown here on the line number four. Inside the return matter we had created a named list. This list has two members. First member is named as total, and we have set its value to total. marks integer vector. While the second member is named as average, which contains average marks. Now let's create this function by executing the lines from one to five. So our function have been created. Now let's use this function. Here on line number seven, we are making a function call. And we have passed two integer vectors as values to the arguments, quiz. marks and vita. marks respectively. Now let's execute this line. And, as you can see on the console, we have two return values represented as two elements of a list. And if you want to get hold of individual result value, you can use any of the list subsetting technique which we have discussed in the data structures module. So, in this clip, we saw how with the help of a list, we can return multiple values from a function. Now, in the next clip, we will discuss another very important concept in functions, that is, functions as objects.

Functions as Objects

In R, functions are first-class R objects just like any other R objects. And because functions are also objects, it means that you can look into them. You can assign them to some other variable. Or you can pass them as arguments to other functions also. So let's take the examples to know more about this. Let's first create a function to work on. Here we are creating a very simple function, GetTotalMarks, which simply calculates the sum of quiz and viva marks and return the total marks. Let's execute these lines from line number one to four. So our function has been created. And if you look into the workspace or the environment tab here on the right, then you will see there is an entry just like we have for any other R object. And because functions are first class objects, we can do various things. And as mentioned into the slides, we can look into them. So now let's see how we can look into them. Remember, to see the content of any variable, we use the variable name itself. Similarly here on the line number seven, we are simply using the function name without any parentheses to look into the function object. Now let's execute this line. So as you can see, we have the whole function on the console. You can also extract different components of this function. formals method on the line number 8 will give all arguments of this function, while body method on the line number 9 will give you the body of this function. Let's execute both lines. So as a result of the first statement, we get different arguments of this function, while the body method returned the function body. Well, other than looking into the function, you can also assign them. Here on line number 12, we are assigning the GetTotalMarks function to a new variable MyGetTotalMarks. Let's execute this line. So we have a new function MyGetTotalMarks which will work similar to the original function. So if you see the content of this

newly created function using the function name as shown here on the line number 13, then you will get the same content. Let's confirm it by executing this line number 13. And as you can see, this function also has the same content. Now let's take a few examples to see how function objects can be used as arguments to other functions also. So traditionally, you call a function like the statement action here on line number 16 where we pass values of different arguments in parentheses. Let's execute this line. So we have the result on the console. After the function execution. So this is how we traditionally call the functions, however in R, there is another way to make function calls. We can use do. call method action here on the line number 19 and 20. In this matter we can pass the function object as a parameter. And for the argument of the function to be called, we can create a list and then pass it to the do. call method as a parameter. So, if you look at this statement, we are passing the function object as an argument to another function do. call. Let's execute the statement. And as you can see, we get the same result to that of the previous statement. So in this clip, we learned that functions are also first class R objects, so you can look into them, assign them and use them as arguments. Now in the next clip we will take our final topic of this module, that is anonymous function

Anonymous Function

So, as the name suggests, anonymous function is a function which has no name. So typically anonymous functions are used when you have to write very small functions. So rather than creating a named function, you can directly use an anonymous function. Let's take an example to see it in action. Here we are in our studio. And we have the same example which we had used in the previous clip. Here we have first created a named function. Named function means a function with a function name. So here we have a function named as GetTotalMarks. Now let's execute these lines here, from line number two to line number five, to create the function. So we have our named function created. Now we can call this function in a traditional manner, or you can use do. call method, as shown here on the line number eight and nine. We have already seen the usage of do. call method in the previous clip. So in this do. call function, we have our named function object as the first parameter. And function arguments in the form of a list as the second parameter. Let's execute this statement. And here in the console we have our results. But if you look closely to this named function, it is a very small function, but even for such a small function we took the pain of creating a named function and then calling it. So in such cases we can use anonymous functions as well. Here in this statement from line number 12 to line number 14 we are again using the do. call method. However, this time instead of the named function object, we have used an anonymous function as the first argument. So the anonymous function can be written in

line. So here we have written the whole function as the first argument, moreover, we have not used the intermediate valuable total. marks. So this anonymous function is literally one liner, and because this anonymous function has only one line in it, we can't remove these curly braces also. But if you have more than one line in and anonymous function then you will need curly braces. So leave the curly braces just like that. And let's execute the statement. And as you can see on the console, we have exactly the same result to that of the previous statement execution. So the anonymous function can be very handy if you're writing small functions.

Summary

So we have reached to the end of this functions module. In this module we first looked at Functions Overview. We learned that how functions help to provide reusability and abstraction. Then we discussed different components of a function. So the function contains arguments, body and possibly a return statement. In R, the final statement is implicitly used as a return statement. After learning different components of a function, we'll look at a couple of naming guidelines for function names. So as per Google R style guide, you can use Pascal casing for function names. Then we'll discuss few important concepts related to arguments. We discussed how arguments are matched if we have multiple arguments and we discussed argument matching by position and by name. Then we learned how default arguments can be used in a function to provide default values. We also discussed the use of ellipsis or triple dots to have additional arguments in a function. Then we learn that R supports lazy evaluation so expression such as, default arguments are evaluated only when they are used for the first time. We also talked about the use of return keyword in conjunction with a list to return multiple values from a function. After learning different concepts related to arguments we discussed that functions are also first class R objects and you can look into them, assign them, and pass them as arguments just like any other R object. Finally, we learned the use of anonymous functions to write small functions without any function name. So this is all about functions. Well, so far in all of our scripts execution of statements have been sequential in. Means we moved from one line to the next line. But R provides several constructs to control the flow of execution depending upon radius conditions. So in the next module we will cover topics related to flow control in R.

R - Flow Control

Introduction

Hi, this is Abhishek Kumar. And welcome to the seventh module on R programming fundamentals. In this module, we will talk about flow control. Well, so far in this course, we have dealt with only those situations where we had to execute one line after another in a sequential order in order to perform some task. But in a real world scenario you may face such situations where you need to control the flow of execution based on certain criteria or condition, or you may have to loop through various elements of an object. Well, in a data analysis project you will encounter such scenarios on a regular basis. So by the end of this module you will learn about various flow control mechanisms such as condition statements, and looping constructs available in R. We will be taking various demos also to further solidify your understanding on these topics.

Outline

Here's the outline of this module. There are three main sections in this module. In the first section we will talk about several conditional statements that are used to write logics based on certain conditions. We will take different conditional statements in R, such as if, if-else, switch, and finally we will look at the vectorized flavor of if-else statement. Then in the second section, we will discuss few looping statements to loop through some values or elements of some object. We will look at various looping mechanisms such as repeat, while, and for. And finally in the last section, we will take you to one more advanced looping mechanism that is apply function, which is a neat way to perform looping in your code. I will also introduce you to other functions in the apply family. So let's start with the first section, where we will first talk about the if statement in the next clip.

If

If statement is probably the easiest technique to incorporate conditional execution in your code. If you have wrote in any other popular programming language, then you must be having an idea of if statement. Well, here is a syntax and typical usage of if statement in your R code. So we've placed a condition in parentheses after the if keyword, and then we used curly braces after the parenthesis. So statements inside the curly braces will be executed only when this condition evaluates to true. The condition in the parenthesis should be evaluated to a logical value, such as true or false. So you can think of condition as a logical vector of length one. Now let's look at one example to see its usage. Here we are in our studio. In this demo, we are creating a function IsGoodPerformanceByBatch, which provides the performance of the batch based on average

marks scored in a classroom test. This function takes one argument, test. marks. Inside this function on line number two, we are first calculating the average of test marks using mean function, and we are also assigning the result to average. marks variable. Then on line number three, we are comparing the average marks with 75 using a logical operator greater than equal to. And we are assigning the output of this comparison to the variable performance. test. So this variable will be a logical vector with one value, true or false, based on average marks. So if average. marks will be greater than equal to 75, then performance. test will be true, otherwise it will be false. Next on line number four, we are also displaying the value of average. marks and performance. test on the console. To do this, we are using the paste method. Paste is a very useful method for formatting the result by concatenating desired strings. The paste method is wrapped by print function so that the output of this line can be displayed on the console. Then on line number five, we are using the if condition. Inside the parenthesis, we have passed the performance. test test logical vector. So if this logical vector will contain true value, then lines between the curly braces will be executed. So lines inside these curly braces will be executed only when the average marks will be greater than equal to 75. Inside the curly braces, we have used a print command to print a statement that overall performance of the batch is brilliant. But if the performance. test variable will contain false value then lines inside the curly braces will be skipped, and the statement at the line number eight will be executed which is a print statement stating that the performance test has been completed. So this is our function implementation. Let's create this function by executing lines from one to nine. So our function has been created. Now let's make some function calls. Here on line number 11 we are passing an integer vector to this function. This integer vector contains four values representing marks obtained by four students in a classroom test. Let's execute the statement. So we have our results on the console. So we can see that the performance. test variable contains true value, because the average of all marks is 77. 5, which is greater than 75. Therefore, the statement inside the if block was executed which was a print statement. And after that the statement after the if block has been executed. Now let's look at one more function call. Here on line number 12 we are passing a different set of four marks to the function. Let's execute this line. If you see the result, we have a false value for our performance. test valuable, because average. marks equals to 58. 75, therefore, statement inside the if blog has been skipped. And the statement after the if block has been executed. So as you can see how with the help of if block, we can control execution flow based on certain condition. So you can take some action when condition for the if block evaluates to true. But if you want to take some other action also, when the condition of the if block evaluates to false, then in that case, you can use the if-else construct which we will discuss in the next clip.

If-Else

With the help of if-else blocks you can execute statements when the condition associated with the if block evaluates to false. It means that the body inside the if block will be executed only when the condition evaluates to true, while the body inside the else block will be executed when this condition evaluates to false. Now let's look at one example to see its usage. In this demo, we have a similar function to that of the previous clip, so I will not go into the details here. However, unlike the function in the previous clip, we have included an else block, also. So if the performance. test variable evaluates to false means that average. marks is less than 75, then we would like to execute statement inside the else block in which we'll print that the overall performance of the batch is average. Now let's create this function by executing lines from one to twelve. So our function is created. Now let's make couple of function calls just like the previous clip. Here on line number 14 we are passing an integer vector containing four test marks. Let's execute this line, and as this time the average is 77. 5, therefore the value of performance. test is true. And only the statement inside the if block has been executed. And after the execution of the if block, the statement after the if else block has been executed. Now let's take another function called where average. marks is less than 75. Here on line number 15 we have another integer vector which we are passing to the function. Let's execute this line. So as you can see in this case, we have an average of 60, which is less that 75, therefore the value of performance. test is false. And this time instead of the if block, the statement inside the else block has been executed. And after the execution of the else block, the statement after the if else block has been executed. So as you can see with the help of if else block you can take action when some condition is satisfied and also when the condition is not satisfied. However, in this case, we've dealt with only one condition that was mentioned inside the parenthesis just after the if key word. But now in the next clip we will learn to use multiple if else blocks if you have multiple conditions and you want to take the decision accordingly.

Multiple If-Else

With the help of multiple if else blocks you can incorporate multiple conditions in your code. Here's the typical structure of multiple if else blocks. So body inside the first if block will be executed only when the condition one will be evaluated to true. Next, rather than only one else condition, we can have other conditions also, so you can put the next condition in else if block. And the body inside this block will be executed only when the condition of this else if block will be evaluated to true. Here in this example, we have only one else if block, but you can have as many else if block as you want. And finally, the statement inside the final else block will be

executed when all previously mentioned conditions will be evaluated to false. So here in this case, if both condition one and condition two will be false, then the body inside the final else black will be executed. Now let's look at one demo to see it in action. In this demo, we have extended our previous clip scenario. Here inside the function IsGoodPerformanceByBatch, we first calculate average marks on line number two using the mean function over test marks. Then on line number three, we are also printing the average value just for reference. Then we have our if block where we want to print the brilliant performance of the badge only when the average marks is greater than equal to 75. Then in the next else if block we have used another condition which is average. marks greater than equal to 60. So it means that first we will check if average marks is greater than 75 or not, and if not then we will check if average marks is greater than 60 or not. And if it is greater than equal to 60 then we will print that the performance of the batch is satisfactory. But if average. marks is even less than 60, then we will move to the final else block where we can print that the performance of the batch is below average. And finally at the line number 13 after the multiple if else blocks, we are printing that the performance test is completed. So now let's create this function by executing lines from one to fourteen. So our function is created. Now let's make few function calls. Here on line number 16, we have our first set of four numbers as test marks. Let's execute this line. So as you can see on the console, because the average marks is greater than 75, therefore, in the result, we have brilliant performance of the batch. So statement inside the if block has been executed and then other else if and else blocks are skipped. And the statement after the multiple if else blocks has been executed. Now let's take the next set of test marks as shown here on the line number 17. Let's execute this line. In this case average. marks is 60, so the condition inside the first if else block will be evaluated to false. Then the next condition at else if block will be checked. And as this condition evaluates to true, therefore, the statement inside this else if block has been executed. And again after this else if block execution, statement after the multiple if else blocks has been executed. Now let's quickly look at the third scenario. Here on line number 18 we have one more set of marks. Let's execute this line. And in this case because the average. marks is below 60, the first two conditions evaluate to false. And therefore a statement inside the else block has been executed, and we have below average performance in the result. So as you can see with the help of multiple if else blocks, you can incorporate multiple conditions in your code. Now in the next clip we will look at another technique that can be used to incorporate multiple conditions in your code. That is the switch statement.

Switch

Switch is also a very useful condition statement where you can incorporate multiple conditions easily. Here's the typical structure of switch statement. We can use the switch function of a level in R. The first parameter to this function is an expression to be evaluated, however this expression should be evaluated to a string or an integer value. Then after the expression, we can provide the actions depending upon evaluated output of the expression. So if the expression evaluates to option one, then body inside the option one block will be executed. Similarly, if the expression evaluates to option two, then the body inside the option two block will be executed. So you can have as many options as you want. You can also have a default value. So if none of the options matches to the expression output, then the default value will be returned. And if you do not provide any default value then NULL will be returned. So this is the typical switch statement structure. Now let's take a demo to see its usage. Here in this demo we are creating a function GetMarksSummary, which takes test. marks as argument, and depending upon the summary type, which is another argument of this function, the function returns the desired result. Here inside the function, we are using the switch function. An output of the switch function is assigned to the result variable. Now let's look at different components of this function. The first parameter is an expression to be evaluated, and we have used the summary. type as the expression. Summary. type is a character string. So depending upon different summary types you will take different actions. So if summary. type evaluates to string mean, then statements inside this block will be executed. Where we are calculating the mean using the mean function in which we have passed test. marks. But if some of your type evaluates to median, then statement inside the second block will be executed, where we are using median function to calculate median over test. marks. Next if summary. type evaluates to character string variance, then the variance will be calculated using var function in R. However, if summary. type does not match to any of mean, median, or variance character strings, then we have set the default value to be not implemented. Moreover, if you look at the switch function, we have only one single statement in each option. Therefore, these curly braces can also be removed. So here from line number 15 to 19, you can see the trimmed down version of the sweet statement where we have removed all curly braces. But if you have multiple lines in any option, then you will have to use curly braces. Finally on line number 20 we have put the variable result as the last statement of the function. So this will be the return value of GetMarksSummary function. So this is our function implementation so let's create this function by executing lines from one to 21. So our function is created, now let's make a few function calls. Here from line number 23 to 26, we have four function calls. For each function call, we are passing an integer vector containing four integer values as the value of the first argument. Then we have set different values for summary type in each function call. So on line number 23, we have used mean character string to get mean of these four test marks. Then on line number 24, we have

used median character string to get median value of these four test marks. On line number 25, we are using variance character string to get the variance of these four test marks. And finally, on line number 26, we have used an unknown character string to test whether the function returns the default value of not implemented or not. Let's test these four lines by executing them. So, all of our function call are executed. Let's look at the results 1 by 1. So in the first function call the string mean was supplied as summary. type variable. And the switch function calculated and returned the mean value which is 77. 5. Similarly, in the second case, the median value is 77. 5 again. The variance is evaluated to approximately 41. 7. And finally, as a result of the final function call where we had parsed the unknown as summary type, the function will not return the implemented character string. So, this was a short example where we have used the switch function to take action based on different values of an expression. Well, so far in this course we have discussed several vectorized operations in different scenarios, such as vectorized arithmetic operations or vectorized logical operations. Such vectorized operations make R a very powerful tool, because they can work on a vector in an efficient manner without requiring any explicit loops. So in the context of flow control, also, we'll be taking one such vectorized statement, that is vectorized If in the next clip.

Vectorized If

Vectorized If is used if you want to work on a logical vector without using any explicit loops. Typically the logical vector can be generated by a logical expression. So to make use of vectorized if in R you can use the ifelse function avalable in R. Here is the typical structure of ifelse function. Inside the ifelse function the first parameter is the test logical vector or any logical expression which can evaluate to a logical vector. The second and the third parameter are two vectors of any type, so if an element inside the test logical vector is true, then corresponding element in the true vector will be returned. While if an element inside the test logical vector is false, then corresponding element in the false vector will be returned. So the length of the true and false vector will be same to that of the test vector. And if true or false vector is shorter in length, then it will be recycled. Means suppose the test vector contains four elements and the true vector has only one element. Then, one element will be repeated three more times internally to create a vector of four elements. Similar thing can be done for false vector also if needed. Moreover, if the logical vector contains missing or NA value, then return value will also be NA. Now let's take a demo to see its usage. Here we are in RStudio. Suppose we have a test. marks vector containing four elements with three integer values and one missing or any value representing marks obtained by four students in a classroom test. Here on line number one we

are creating an integer vector using a combined function. Let's execute this line. So our integer vector is created. Next on line number two, we have a logical expression. Where we are using the logical operator greater than equal to, to compare test. marks with the value 75. Let's execute the statement. So as you can see, we have a logical vector with four values. We have NA also corresponding to the last element in the test. marks vector. So as you can see, with the help of a logical expression, we can generate a logical vector. So we can work on such logical expression. So next, on the line number three and four, we are using the ifelse function. The first parameter is the logical expression, same as the previous line. So this will return a logical vector with four values. Then the second and the third parameter are vectors with four values each. The first vector contains four character strings, which will be returned if elements inside the test vector will be true. The second vector, again, contains four character strings, which will be returned if elements inside the test vector will be false. So let's execute this statement. Let's examine the result. As the first value in the test vector was false, therefore the first element from the false vector was returned. And as the second and the third element in the test vector were true, therefore the second and the third element from the true vector were returned. And finally, as the fourth element in the test vector was any, therefore we have any in the result at the false position. So as you can see, due to the vectorized nature of the ifelse function, we worked on different elements of a logical vector, generated by a logical expression without using explicit loops. Now, let's take one more example where we have fewer number of elements in the true and false vector. In fact, here on line number five, we have only one element each in the true and the false vector. So, in this case, these true and false vector will be recycled. So, you can take this true vector as a vector having four elements, where each element is the same having the value of with distinction. Similar case will be there for the false vector also. So, let's execute the statement and here's the result. So, if the value falls in the test vector, then without distinction is returned. While if value is true, then with distinction is returned. And for NA in the test vector the return value is NA. So, this is the benefit of vectorized operations. Because they help you to write code neatly and efficiently without writing explicit loops over the different elements of a logical vector. But many times, you will run into situations where you will have to write explicit loops. Therefore, from the next clip, we'll be taking few basic looping mechanisms available in R starting with the repeat loop in the next clip.

Repeat

Repeat statement can be used for creating loops in R. Here is the typical structure of the repeat statement in R. We use repeat keyword, then inside the curly braces you can place your

statements which will be executed in a loop. However, in this case execution will be performed forever, and the loop will go on and on, because there are no stopping conditions. Now let's take a simple demo where this repeat statement is used. Here we are in RStudio. Let's take a fictitious scenario. Suppose a teacher ask a student to write on each page of a notebook. So for that we have created a function write on notebook. Inside the function on line number three we have first created a variable count which will refer to, the page number or page count. We have set its value to 0. Then we have our repeat loop. In the body of the repeat loop, we first increment the value of count by one. And then we have used print function to print the page number on which we are writing. So this is a very simple function. Let's create this function by executing lines from one through eight. So our function is created. Next on line number ten we are invoking this function. Let's execute this line. So as you can see the statement inside the repeat loop will be executed forever. And until and unless you click this red stop button here or you close the RStudio itself. Let's stop it now. So as you can see you should use this structure of repeat only when you want to execute something for eternity. However, in most of the real world situations we will provide a stark condition to break the slope at a desired point. So let's take that scenario in the next clip.

Repeat With Break

Typically we use repeat in conjunction with a break statement. So inside the body of the repeat block, we can use an if statement with a loop stopping condition. So if the condition associated with the if statement evaluates to true, then we can take some action and finally can use break statement to stop the loop. So break is the keyword to exit the loop. Now let's take the previous clip demo and incorporate some breaking mechanism to stop the infinite loop. Here we are in RStudio, and we have made some changes in the WriteOnNoteBook function discussed in the previous clip. Here we have used one argument that is total. page. count, which represents the total number of pages on the notebook. So, we want to stop writing on the notebook when all pages of the notebook are finished. So this is our stop condition. To incorporate this condition, we are using an if block inside the repeat loop. We are comparing the count variable with the total. page. count variable. And if the value of the count exceeds total. page. count then we will stop the loop. So inside this if block first we print back the pages have been finished and then we have used break statement to exit from the repeat loop. Let's create this function by executing lines from one to 12. So the function is created, let's make a function call now. Here on line number 14 we have passed the value of 10 as total. page. count, let's execute this line. So as you can see the loop has been stopped after 10 number of iterations, because the value of count exceeded the total. page. count value after that. So this is how you can put your desired condition in the if block

and use a break statement to stop the loop at the desired moment. But, suppose along with the break condition if you want to skip through loops based on certain conditions, then you can use the next keyword. So in the next clip we will see a combination of repeat, break, and next.

Repeat With Next

Repeat with next can be used to skip loops depending upon certain conditions. So inside the repeat loop we have again used an if block with a certain condition. And inside this if block we can use the next keyword. So whenever this statement with the next keyword will be hit, then the current loop will be skipped from that point and the next loop will be started. Now let's take a demo to see its usage. Here we are in R Studio, and we are extending the previous demo. Suppose we want to incorporate some logic in the repeat loop so that we can write only on odd numbered pages. Means all even number pages should be skipped. So for that we are using another if block here with the next keyword. So as the condition of this if block, we are evaluating the value of count whether it is an even number or not. So for that we are calculating the remainder value, when divided by 2, using the modulus operator. And if the resulting value will be 0, then it means that the value of count is divisible by 2, or it is even. So for that condition, we are first painting that, we are skipping that page number. And then using the next keyword we are skipping the current loop. Means the statement after this if block will be skipped, and execution will be moved to the start of the repeat loop. So let's create this function by executing the lines from 1 to 16. So our function has been created. Now let's test it by making this function call at line number 18. Let's execute this line. So as you can see in the result, the even numbered pages are scaled while only the odd numbered pages have been written. And finally, due to the break condition, the loop stopped and the value of count leads to 10. So, with the help of break and next keywords, you can control the execution of your repeat loop based on desired conditions. However, one point to note is that, for all repeat cases the conditions are checked once you are in the repeat loop. So repeat loop are just like do while loops, if you have worked in other popular programming laguages. But, if you want to enter the loop only when sudden conditions are met, then you can use the while loop which we will cover next.

While

While loops are also a very popular looping mechanism in R. While looping are very similar to repeat with break loops. However, in the case of while loop the conditions are checked before entering the loop. Here's the typical structure of the while loop. The condition inside the

parentheses is checked first, and the body will be executed as long as the condition associated with the while loop evaluates to true. Means, once the first loop or first titration is finished, the condition will be checked again and if it evaluates to true, then the body inside the while loop will be executed once again. So the iterations will continue as long as the condition inside the parenthesis remains true. Now let's take a simple demo to see the while loop in action. In this demo, we have created the same function writeon notebook, which we have discussed in the previous few clips. However, in this demo, we have replaced the repeat with break loop, with a while loop. So here as the while loop condition, we are comparing the count variable with total number of pages. So the statement inside this while loop will be executed til this condition will hold true. Next inside the while loop we are first incrementing the value of count by 1. And then we are using the print function to write on the page. So once this line will be executed then again the condition in the while loop will be checked. And if it will be true, then again the statement inside the while block will be executed. This loop will continue till the condition becomes false. And that will happen only when the count value will go past the total number of pages. And at that moment, the loop will be stop. And the statement after the while loop will be executed. So, now, let's create this function by executing the lines from one to nine. So our function is created. Now, let's make a function call at line number 11 to test the working of this. Next let's execute this length. So as you can see in the console, the statement inside the while loop has been executed for 10 number of times. Every time the value of count has been incremented by one and when the value of count has at least the value of 10, then after that the value is stop. And the statement after the while loop is executed, which says that the pages on the notebook have been finished. So this is how you typically use the the while loop in your code. In the next clip, we will look at another very popular looping construct, that is for loop, which can be used to loop through a vector.

For

If you have worked in any other programming language, then you must have seen or used for loops in your logic. In our typically for loops are used to loop while I tread through a vector. That vector can be of any type. Here is the typical structure of for loop in R. Inside the parentheses of for statement we have an iterator variable and that will loop through different elements of the vector one by one. So the number of times statement inside this for loop will be executed will be determined by the number of elements in the vector. Now let's take a simple demo to see the usage of for loop. Here we are in our studio and we have the same write on notebook function. Which we have seen in the previous clips also. However, this time instead of the while loop, we

are using the fall loop. Let's look at the structure of this fall loop. So inside the parenthesis we have used count as our iterator variable, and we are looping through an integer vector. The integer vector has been formed using a sequence operator. So here we are creating an integer vector which contains integers from 1 to number of pages in the notebook. And inside the for loop, we have our print statement where we are printing the page number of the notebook on which we'll be writing. So once the for loop will be finished, then the statement after the for loop will be executed. Now let's create this function by executing the lines from one to seven. So our function has been created. Now let's make a function call by executing this line number 9. So as you can see in the console the statement inside the for loop has be executed ten times as the vector inside the for loop parenthesis contains ten integer elements ranging from 1 to 10. So this is how you can use for loops in your logic to iterate two different elements of a vector 1 by 1. Well, so far in this module, we have talked about several basic flow and total techniques, including various conditional statements and looping constructs. Now in the next clip, I will introduce you to one more advanced family of looping mechanisms, that is, apply family of functions. As these are more advanced topics, therefore I will not be going into the details in this fundamental course. However, you will get an overall idea.

Apply

Apply family of functions are more advanced flow control mechanisms in art. Which we will cover in our advanced level course on art. However, in this fundamental course I will introduce you to the basic Apply function. Apply function is vitally used in data analysis projects also. So what is the Apply function? Well if you look at its definition in the r documentation then it says that the apply function returns a vector or array or list. Obtained by applying a function to margins of an array or matrix. Oh, that was one lengthy and complex sentence. Let's break it up. So the Apply function is meant to work on margins of an array or matrix. So what do we mean by margins here? You can think of margins as either rows or columns or both. Both means each element of the array or matrix. So we can apply any function on rows or on columns or on individual elements of an array or a matrix. And the result can be a vector or an array or a list given any point the usage of Apply function. Here's the typical structure of the apply function. So we can pass the data as our first argument. This can be an array or a matrix. Then we can pass the margin information as the second argument. So for rows it would be 1. For columns it would be 2 and for both Rows in columns, we can use 1 colon 2. Finally the last argument on our function which we want to apply on margins of data. So this is the general structure of the apply function. Now let's take demo in the next clip to know, how apply function can help you and how it can be used.

Demo: Apply - Part 1

Here, we are in our studio. In this demo, we'll be working on a matrix. So let's first create a matrix to work on. I will not go into the details of creating a matrix as I have already covered it in detail in the data structure module. So if you are not sure about creating matrix, then you can watch the matrix clip available in the data structure module. So here from line number 1 to line number 8, we're creating a matrix, student. marks, that contain marks obtained by 4 students in 5 classroom test. We are using the matrix function to create the student. marks matrix. The first parameter is an integer vector containing marks obtained by four students in five subjects. So these are 20 integer values. We have set the number of columns to be five and number of rows to be four. You also explicitly set by loop parameter to 2 so, that the values of this integer vector are arranged row-wise, and making it a 4 by 5 matrix. Next we have set the row names and column names as well. So, we have used four student names to name rows, and five subject names to name columns. Let's create our matrix by executing lines from one to eight. So our matrix has been created. Let's look at this newly created matrix using the matrix variable named action, here on the line number 9. So here is our matrix with four rules. Each rule representing marks obtained by each student. And each column represents math scores in that subject by four students. Now suppose you want to total obtained by each student in all subjects. Then what will you normally do? One such solution could be using two nested for loops as shown here from line number 10 to line number 18. So first we can create an empty vector to store the total for each student then the first for loop can iterate through the number of rows of student. marks matrix. And inside this first for loop we are first setting the variables sample 0 and then we have a nested for loop that iterates through the number of columns in the matrix. And inside this nested for loop we are adding the cell value to the sum variable. This inner for loop will return total marks obtained by one student. So after this inner for loop we can assign the sum total to the result vector using proper indexing. And once you are done with both for loops, then you can print the result using the result variable. So let's execute these lines, from line number 10 to line number 18, to get the desired result using for loops. So here is our result. We have four values in the result. Each representing the sum of marks obtained by each of the four students. So as you can see it took us seven lines to achieve the desired result. However, with the help of apply function, you can do it in a single line without writing explicit loops.

Demo: Apply - Part 2

Here on line number 21, we have used the apply function, and inside the apply function the first parameter is the matrix student. marks. For the second parameter I have used the value of 1 as

margin. That means we want to work on rows. And finally the last parameter is our function. Here we have used the sum function, as we want to get the row-wise sum. Let's execute this line to see the result. And here on the console we have row by sum in the form of a vector containing four values, where each value represents the sum of marks obtained by a student. So as you can see, how with the help of a single line, we are able to accomplish the same task, which took us seven lines with two for loops. So apply function provides a very neat way to perform looping. Not only this, apply function can give a better performance over badly written for loops. Now let's see few more examples of the apply function usage. Suppose you want to get maximum marks scored by each student, then you can use the max function as shown here on the line number 22. Let's execute this line. And here on the console we have our result. But if you want to know in which subject the student has scored most, then you can use another very useful function that is which. max. As shown here on the line number 23. Let's execute this line. So here on the console we have index numbers or column numbers in the result. So if you want to know the subject name itself instead of these index numbers, then you can use subsetting technique on column names as shown here on the line number 24. So we are using the index values to get corresponding subject names from call names character vector. Let's execute this line. And here in the console we have subject listed in which corresponding student has scored the most. So in these four examples we have logged row-wise, as we have used one as the margin parameter. Now let's take a couple of examples with two as margin parameter. Two means we want to work column wise. So suppose you want to get average scored in each subject, then you can use two as margin parameter. So here on the line number 25, we have used two as the second parameter and we have used the mean as the function. Let's execute this line. On the console, we have average marks scored in each subject. Now suppose you want to know the maximum marks scored in each subject, then you can apply the max function on student. marks columnwise as shown here on the line number 26. Lets execute this line. And here on the console, we have our results. Now if you want to know which student has all the maximum marks in each subject, then, you can use the which. max function column wise. However, here we'll be using the subsetting on rownames vector as shown here on the line number 27. Let's execute this line, and on the console we have the name of students who have gotten maximum marks in the respective subject. Next on line number 28 we have used 1:2 as a margin parameter. Means we want to work on each element of the matrix. So suppose we want to add two marks in each student marks in each subject, then we can write something like this. Here at the last parameter we have used and, and anonymous function that will take each set of value of the matrix and will add a value of two. Let's execute this line and here is our result pair two value has been added to each set. So in this demo we saw the usage of applied function, and how it can be used to perform looping neatly. Applied function may give

you performance benefits also in many cases. But this Apply function is typically on an array or a matrix, and it returns a vector or a list or an array. However, there are other functions also, in that apply family of functions, which can work on other data structures as well and can return results in a different format. But as this a fundamental course, I will not go into the details of other functions of the apply family. However, let's quickly look at the other functions of the apply family.

Functions in Apply Family

There are L apply, S apply, and V apply functions in the apply family, which work on a list or a vector. While L apply can return a list, S apply and V apply can return a vector, a matrix, or even an array. There are some other apply family functions also used in special cases, such as R apply which is a recursive version of L apply, while M apply is the multivariate version of S apply function. We'll be taking these advanced looping techniques in an upcoming advanced level course on R. But if you want to know more about these functions, then you can always refer to the R documentation.

Summary

So, we have reached the end of this module. In this module, we discussed various flow control mechanisms available in R. We started with conditional statements that can help us to take appropriate actions based on certain conditions. We first looked at the if statement so the statement inside the if block will be executed only when the condition associated with the if block evaluates to true. Then we learned the use of If-else, where else block can be used to execute statements if condition of the if block evaluates to false. We also discussed the use of Multiple If-else blocks to incorporate multiple conditions in the code. Then we learned the use of switch function to act as per different values of a string or integer expression. And finally, we looked at the vectorized version of if using if-else function that can work on vector of expressions. After learning these conditional statements, we looked at few basic looping techniques, starting with repeat. Repeat can be used to execute the call in a repetitive manner. You can also use break statement inside an if block to provide start condition of the loop. We have also learned the use of next statement to skip loops based on certain conditions. After repeat loops we learn another looping technique, while, that can provide a looping mechanism til the condition associated with the while loop remains true. Then we learn the use of the popular for loop in which we can use an iterator variable to iterate our vector. If required you can use multiple nested for loops also to accomplish different tasks. After learning some basic looping techniques. I introduce to you to

one advanced looping technique that is apply function. Apply function can help to implement a loop in a very neat fashion. You can use the apply function on an or a matrix. You can either work row-wise, or column-wise, or on individual elements. Finally, I also listed out some other functions in the apply family, which we will be covering in an upcoming advanced level course. So this was all about full control in R. Well so far in this course we have worked with base R framework with standard packages only. But once you will start using R in your projects you may have to use other add on packages also. There are thousands of add-on packages I've labeled for R environment that can help you to accomplish various tasks. And because packages are an integral part of our framework we'll be discussing various nitty gritties associated with packages in the next module where you will learn to install. Load and manage add-on packages. So, see you in the next module

R - Packages

Introduction

Hi, this is Abhishek Kumar, and welcome to the eighth module on R programming fundamentals, which is on R-Packages. So far in this course, we have worked mainly with the code R framework. So when you install R for the first time on your machine, the code R framework will be installed along with some standard packages. However, there are thousands of extra add on packages at your disposal, meant for different types of tasks. R-Packages are one of the most compelling features of R framework, as these packages provide ready-made solutions to its users, for different types of real world problems. So this module is completely dedicated to R-Packages. By the end of this module, you will not only learn some fundamental concepts related to packages, but also to install them, use them, and manage them efficiently.

Outline

Here's the outline of this module. We will first discuss some fundamentals of R-Packages. Then we will learn to load packages, which are already installed on your machine. Then we will discuss the process of installing new packages on your machine. And finally, we'll discuss techniques to maintain and upgrade the installed packages. So let's start with some fundamentals on R-Packages.

About R Package

So what is an R-Package? Well, an R-Package is basically a collection of different items, such as, Functions, Datasets, Compiled code and Documentation. So typically, each package is intended for a certain area and are used to accomplish certain tasks in that area. These packages are available in some central repositories. From there, anyone can download these packages. So when you download and install an R-Package on your local machine, then contents of that package is copied from the repository and stored in a directory on your local machine. That directory is referred as Library in R. So let's switch to a demo, to know more about R-Packages. Here we are in RStudio, so if you want to see a list of all installed packages on your machine, then you can use the library command, as shown here on the line number 1. Let's execute this line, as you can see the results are displayed in another tab in RStudio. Here you can see, there are several packages already installed on your machine. These packages are installed when you install the base R version on your machine. This list also contain the add on SOS package, which we had installed explicitly in the second module. Moreover, if you look at this line which says, packages in library followed by some folder path, so we have few packages installed at this location, while few other packages are installed at another location. Let's look at this location, let's go to this folder path in the explorer. As you can see, we have separate directories for each package, in this folder. Each directory is also referred, as library of that package. Like here, we have this cluster folder. So when this package was installed all items available in the cluster package, was simply copied into this cluster directory, which we refer as cluster library. Let's look into this library by opening this folder. Inside the cluster library we have different types of items, that are again organized in different folders. So we have some help files and documentation in the help folder. We have some other items also in this folder, containing some functions and code snippets. Some R libraries may also contain data sets as well. Coming back to RStudio, let's look at another command, that can be used to see all installed packages. This command is installed. packages command. This command will give you some other extra information also. Here on line number 2 we have assigned the result of installed. packages command to a variable packages, using an assignment operator. Now let's execute this line. So this line has been executed. And if you see in the workspace tab on the right, we have an entry for the newly created variable packages. Let's click this small grid icon, to see the content of this variable. This variable is a character matrix. Here we have different columns, and each row corresponds to one installed package. So here, you can see various information related to installed packages, such as package name, library path, its version, various dependencies. There are other informations also. So, installed. packages command will give you a comprehensive list of all installed packages, with some important information. Well,

this was a brief introduction to R-Packages, but remember, just by installing an add-on package does not mean, that you can start using that package just like that. Because, if you want to use any add-on package, then that package has to be loaded in the memory first. So let's learn about the loading process in the next clip.

Load R Package

So what do we mean by loading a package? Well, loading a package means, loading the package in your computer memory, and to make it ready to use. So until and unless a package is loaded, we cannot use any function or item, available in that package. Therefore it you want to use any item of a package, then we need to load and attach that package explicitly in the current R session. But you may be wondering, why explicit package loading is required in the first place. Why R does not load all installed packages, when we start any R session? Well, R does not load all installed packages by default purposefully. When you install R on your local machine, only few standard packages are installed. But over the period, you can install various add-on packages. And after some time, there may be a large number of packages installed on your machine. And if R will load all packages at the start of the R session, then not only it will take more time to start the R session, but also, it will eat up your memory unnecessarily. So to avoid such situations, add-on packages are loaded into the memory, as and when required. Now let's take a demo, to learn more about this whole process of loading and unloading of packages.

Demo: Load R Package - Part 1

Here we are in RStudio. So, to see the list of all packages which are currently loaded and attached in R session, you can use the search command action, here in the line number 1. Loaded means the package is loaded into the memory. And attached means the package is attached to the current R session, and can be accessed via the search command. Let's execute this line. So here we have the result on the console. You can see a few package names also in the result. These packages are loaded into the R session by default, whenever you launch the RStudio. So whenever you use any function or object, then R searches the function or object, in all packages that are currently attached to the search path. So later if you load and attach any another package, in the current R session, then that package name will also be listed out here in the search command response. Now suppose you want to use some functions, available in the parallel package, then you have to load the package into the memory and to do so you can use the library command. And in the library command, you can parse the package name as an argument. You

can wrap the package name in quotes, as shown here in the line number 4, but that is not mandatory. You can also use the package name without quotes, as shown here on the line number 5. However, make sure you are using the correct casing for the package name. Now let's execute this line number 5 to load the parallel package. So, the parallel package is loaded into the memory, and if you use the search command again, then the parallel package should also be included in the result. Let's confirm it by executing this statement on line number 6. And as you can see, the parallel package is attached to the search path, and is listed out here in the result. So this was about attaching or loading a package, but you can also detach a package, if that package is no longer required in the current session. For that you can use the detach command, as shown here on line number 9. Inside the detach function, you can pass a package name like this. So you can use the package keyword with a colon sign, followed by the package name to be detached. We have set another field, unload, to be true. The default value of unload field is false. By setting this field to be true, we are making sure, that the package is no more attached as well as, is no more loaded into the memory. If you use the default setting of false, then that package will be detached from the search path, but will remain loaded into the memory for the current R session. So now, let's execute this line. So, this command has been executed, and the parallel package is unloaded and detached from the current R session. So if you use the search command again, to list out attached packages, then the parallel package entry will not be there. Well, so far we have looked at commands to attach and detach a package programmatically. But you can do it by the RStudio GUI also. Here on the Packages tab, you can use these check boxes, to attach or detach a package. So if the check box is unchecked, then that means that the package is not loaded, and you can click the check box to load that package. So the package will be loaded. And if you uncheck the check box, then that package will be unloaded. So in this manner you can also use the RStudio GUI, to attach or detach a package. Well, so far we have loaded or unloaded only those packages, that are already installed on your machine. But if you tried to load a package using the library command that is not installed on your machine, then, you will get an error.

Demo: Load R Package - Part 2

Here, on line number 12, we are using the library function, to attach a dummy package named as newpackage. But this package is not installed on this local machine. Now let's execute this line. As you can see, we get an error message, that there is no such package installed, which can attached or loaded. So in a real world scenario, if you are not sure whether the package is installed or not, before loading the package, then you can use another R command to load a package that is require command, as shown here on the line number 14. So typically we use the require function,

in an if block, as shown here from line number 14 to 17. This require function returns true if the package is already installed. And the package will be loaded and attached. But if the package is not installed, then it will return false. And rather than raising an error, a warning will be issued, that the package is not available to be loaded. Here we have used not logical operator, before require function. So the code inside the if block will be executed only when the package is not installed on your machine. And in that case, you can simply write a command, to install the package in the if block. Let's execute this if block. As you can see, we have got no error, rather than a warning message, that the package is not installed on your machine. So inside the if block, you can write a statement to install and load the required package. So now in the next clip, we will talk about package installation. Where you will learn to install packages, from various repositories.

Install R Package

To install a package, you need to first download the package, from some repository. So a repository can contain several packages, and you can download and install the required package from the repository. We have already discussed one such repository in the introductory module, which is CRAN, or Comprehensive R Archive Network. The repository has currently more than 5, 500 packages as on date. And more and more packages are getting added to this repository. These packages are developed, for a variety of application areas. To know more about different areas, you can visit the CRAN Task Views link. Here packages are segregated area-wise. So on the right hand side, we have application area description, and on the left hand side, you will find the hyperlinks. These hyperlinks will take you to the webpages, which contain details about related R-Packages, available in CRAN. So as you can see here, coverage of this repository is very wide. So we have packages related to Econometrics, Ecology, Finance, Genetics, MachineLearning, NaturalLanguageProcessing, Optimization and many more. So you can look into your relevant area and download and install the required package, from the CRAN repository. CRAN repositories are available worldwide, through mirrors. These mirrors contain identical packages and these are geographically scattered across the globe. So that, you can download the package from the mirror nearest to your place. Here's the link for the available CRAN mirrors. You can check this link to see different CRAN mirrors. CRAN is a great repository for R-Packages, in various application areas. But CRAN repository is not the only repository for R-Packages. There are other repositories as well, such as BioConductor. This repository typically contains packages, related to bioinformatics. It has more than 800 packages, available as on date. Other than BioConductor, you can use Omegahat repository, or R-Forge, and rforge. net repository. R-Forge

and rforge. net repository, typically contain developing version of packages, that are added to the CRAN repositories eventually. Other than these repositories, there are other online repositories also, and one such repository is github, which is very popular among developers across the globe. Well, so far we have looked at different repositories. Now let's look at a demo to see, how to actually download and install a package, from a repository.

Demo: Install R Package

Here we are in RStudio. You can install various packages using RStudio GUI, or through the script. Let's look at the first technique that is using the RStudio GUI. So to install a package using the RStudio GUI you can go to Tools, then go to Install Packages. So if you are doing it for the first time, it may ask for the CRAN mirror. So select the CRAN mirror accordingly, and after that you will get this dialogue box, for package installation. Head in the first combo box you can see the repository from which you can download the package. Currently it is set to CRAN, but you can set the repositories, as per your requirement. We will see this in just a moment. So below the combo box for the repository, we have the test box, where you can write the package name. If you want to install multiple packages at once, then you can separate the package names, with space or comma. Then you can also see the library location, where the package will be installed. The checkbox here will help you, to install all dependencies of the packages as well. Now let's install a package, named as ggplot2, which is a very powerful graphics library in R. Now let's click Install. So the command has been issued on the console, to install the package. The package will be downloaded from the repository, and finally installed into the library on your local machine. So as you can see, the ggplot2 package has been installed, along with all of its dependencies. Now suppose you want to add more repositories. Then you can use the setRepositories command, as shown here on the line number 1. Let's execute this line. On the console you can see, we have different options here. You can set the desired repository as per your requirement, or you can have multiple repositories as well. So if you want to add BioConductor software repository along with CRAN and CRAN extra repository, then you can write 1, 2 and 6 separated by spaces. Now hit Enter, to send the repositories. So now you can download packages from all three repositories. In fact, if you will come to Tools, then go to Install Packages, then you will see the BioConductor software also in the first combo box. And now again, you can type here the desired package in the text box, to download and install from any of these repositories. So this was about installing a package from GUI. But you can do it programatically also. So coming back to the script, here on line number 2, we are using the command install. packages, and inside this command, we are passing the package name, wrapped in quotes as an argument. So this statement will install one

package, but if you want to install multiple packages then you can again use the combined function, as shown here on the line number 3. Here we have written three packages names, wrapped in quotes inside the combine function. You may include as many package names as you want. Let's execute this line number 3. So as you can see all these packages are installed to the default library location. But you can set the library path explicitly also. To know more about this install. packages command, you can use the inbuilt documentation. Now let's see, how can you download and install a package from github repository? So to download and install a package from github, you can use the install_github function as shown here on the line number 6. Here you can pass the package name wrapped in quotes as the first argument. You can also pass the github user name of the author, wrapped in quotes as the second parameter. Here on this line, I'm installing one of my favorite package Slidify, developed by Ramnath Vaidyanathan. Slidify helps to create stunning presentation in HTML file, by converting the R mod down. So this statement will download and install the Slidify package, on your machine. However, to use, install_github function, you need to install and load another package named as devtools, which is available through the CRAN. So here on line number 4, we are first using the install. packages command to download and install devtools package. Then on line number 5, we are loading the devtools package. So let's execute line number 4, 5 and 6 at once, to first install devtools, then load devtools. And then finally install Slidify, on your machine. Let's execute these lines. So the devtools package has been installed. Let's load it. Now let's install the Slidify package. So the Slidify package, has been installed on your machine. So in this clip, we looked at techniques to install packages from various repositories. You can use the GUI as well as the script to do so. Well, as you will start working with R framework, there will be so many packages installed on your machine over the period. So it is very important to manage the installed packages efficiently. So in the next clip, we will look at a few commands and techniques, to maintain various installed packages on your machine.

Manage R Package

When you start working on various R based projects, you will be installing a number of R packages on your machine. And over the period, this number can become very high. So you will have to maintain, these installed R packages. The maintenance include, updating existing packages, as well as, removal of packages that are no more required. So let's look at few useful commands and techniques, which can help you to maintain various installed packages, easily and efficiently. Here we are in RStudio. Let's first learn to maintain packages from the RStudio GUI itself, then we will look at commands to do it programmatically. In RStudio, you can use packages

tab to maintain installed packages. Let's first look at this process, to update install packages. So on the packages tab you would find this, check for updates button, click this button. This will launch a dialogue box, but if you have it installed, all packages, then you may get a message that all packages are already up to date, and there is nothing to update. In fact, I have deliberately installed older versions of ggplot2 and sos package, just for this demo. But over the period you will get packages updates, and if there are any update, available that will be listed out here, in this dialogue box. So here you will see the package name, installed version, available version, and link to some package-related news. You can select one or more packages from this list, and update them in the latest version. Let's cancel it for now, because we will do it programmatically. Now if you want to remove any package from your machine, then you can click these cross items. This will remove the package permanently from your machine. So in this manner, you can use the RStudio GUI, to easily maintain, installed packages. Now let's look at few R commands to perform the maintenance task, programatically. The R command to update packages is update. packages, as shown here on the line number 1. Let's execute this line. So if you use this statement, then it will prompt for confirmation to update each package, for which update is available. This may be an issue if you have larger number of packages to be updated. Let's cancel this process by typing C and hit Enter. So the alternative approach is to set the parameter value to False for the ask parameter, as shown here on the line number 2. This will not prompt for any confirmation, and will update all packages, for which updates are available. Let's execute this line. So as you can see each package is updated, for which update is available. So this command can be very helpful to you, to keep your R packages up to date. Next, if you want to remove any package programmatically, then you can use the remove. packages command, as shown here on the line number 3. You can pass the package name wrapped in quotes as a parameter to this command, but if you want to remove multiple packages in a single line, you can use the combine function as shown here on the line number 4. You can pass multiple function names, wrapped in quotes to the combine function. Let's use this line number 4. So the removal process has successfully completed. So this was all about maintaining R installed packages. You can update and remove installed packages through GUI, as well as programmatically.

Summary

So, we have reached to the end of this module. In this module, we started our journey, by learning some important details, about R-Packages. So R-Packages typically contain functions, datasets, compiled codes, and documentation. These R-Packages can be downloaded from various repositories. One of the most famous R repository is CRAN, or Comprehensive R Archives

Network, which has more than 5, 000 packages as on date. We looked at some other repositories as well, such as BioConductor, R-Forge and GitHub. We also learned that, when we install a package, then that package is installed in a directory on a local machine, which is also called as the library. And then we learned that, if we want to use any add-on package, then we have to load it first in the memory, using library command. Then in the third section, we learned techniques, to install R-Packages. We learned how to install packages through our GUI, as well as programmatically using install. packages command. This command can be used to download and install packages, from various other repositories. We also learned the use of installing_GitHub function, if you want to download and install a package, from the popular repository GitHub. Then, in the final section, we learned to manage install packages, so we can update packages, to their latest versions, using update. packages function. You can also use remove. packages function, to remove a package, if that package is no more required. So in this module, we learned various aspects of R-Packages. This module is also the last module of the Core Programming section. So in the Core Programming section, we learn various constructs of R language. From data structure to functions, to flow control, and finally packages. Concepts learned in the Core Programming section, will be very helpful to you in your journey with R. Now in the last section, we will learn some basics of working with data in R, in next couple of modules. For any data analytics project, you need to have data first. So in the next module, we will learn how to get data, from different types of sources. So see you in the next module.

R - Import Data

Introduction

Hi, this is Abhishek Kumar. And welcome to the ninth module on R Programming Fundamentals, which is on importing data in R. This is also the first module in the data analysis section. For any data analysis project using R, we first need to bring all required data in the R environment so that we can work on them. In the real world scenario data may be available from a variety of sources and in a variety of formats. So by the end of this module you will learn to import data from some of the very common and popular data formats and data sources.

Outline

Here's the module outline. There are three main sections in this module. In the first section, we will learn to import different types of local and remote files in a variety of formats. We will take CSV files, generic table files, then XML files, and finally, Excel files. Then in the second section, we will learn to import data from built-in data sets which are available in various R packages. And finally in the last section, we will talk about R support to import data from variety of databases. So we will start by importing local and remote files. But before we start working with any file, let's first discuss one very important concept, that is working directory. Working directory will help you to understand the location of the desired files on your file system. So let's learn more about working directory in the next clip.

Working Directory

Working directory is a very common and important concept if you have worked in any of the language environment. Well, in the context of the data import and export the working directory is the default location for the file for which it can be read by the R. And it is also the default location of the file where it will be exported. So until now unless you specify some explicit path to work, working directory will come into the picture. Therefore you need to make sure that you are working in the correct directory, otherwise you can run into problems. So, in R you need to know a couple of commands. First is the getwd command, which is used to get the current working directory. So you can use this command to check the directory in which you're working. The second command is setwd command. That can be used to set the working directory as per your requirement. Now let's take a quick demo to know more about working directory.

Demo: Working Directory

Here we are in our studio. On line number two we are using the getwd method to get the current working data tree. Let's execute this line. So we have our current working data tree on the console. You can have a different output on your machine depending on your R installation or your previous work in the R environment. Now let's see how we can set the working data to as per your requirement. Before looking into the views of setwd in the script, I'll show you one very easy way to set the working directory using the RStudio interface. So you can go to the Session menu on the top and then go to Set Working Directory and then you can choose the directory by clicking the Choose Directory menu. This will launch a folder dialog box. And then you can choose your desired path. Let's cancel it for now because here in this demo, we will set the working directory from our R script. So here on line number 3, 4, and 5, we have passed an absolute part

of the desired location to the setwd command. You can set the part as per your requirement on your machine. However, if you look at the separators, then in all three cases you will find different separators. So different separators will be used on different operating systems. So, on Windows operating system you can either use the double back slash or the single forward slash, while on Mac and Linux you have to use single back slash as separators. So as you can see, if you are passing the absolute part in this manner, then you will have to remember the separator for the corresponding operating system. However, there is another way which is platform independent. Here on line number six we are using the file. path method. Inside this method we can cascade drive letters and folder names easily. This method will figure out the separator by itself. So inside the file. path method we have put our drive letter and folder names in the correct order. And finally, we have passed the output of the file. path method to the setwd command. In this module, I will be using the file. path method meantioned on the line number 6 to set the working directory for the R sessions on my windows machine. But you can set the working directory asked by you in admin. So as now, you have the understanding of working directory, we can move further to import data from various external files, starting with the CSU file in the next clip.

Import CSV Files

CSV, or comma separated value files, are one of the most common file formats for storing data. So if you have opened a CSV file in a Notepad application, then you will find values are separated by comma. So you may have header information in the file, and then in each row values will be stored and are separated by comma. Each row in the file represents one data instance, while each column represents a feature or attribute. You can also open the CSV file in a spreadsheet application such as Excel where values will be arranged in different cells of the spreadsheet. To import CSV files in R you can use the read. csv function. This function takes several parameters. Let's talk about a few important parameters. First is the file name. File name is the required argument in the read. csv function. So you directly pass the file name to the read. csv function to import data anal. In this read. csv function the header parameter is set to true by default. So if you do not have headers in your CSV file, then you will have to set this value to false. More over the default value of sep, or separated parameters is comma, but if you have some other character as separator then you can set this value accordingly. Well, if you'll use read. csv function to import the file in R, then values will be stored in data frame. For example, here are the execution of statement, the my. data variable will be contain values in the form of a data frame. Now let's take an example to see the usage of this function.

Demo: Import CSV Files

For this module we have placed all the required files in the data folder. You can download scripts as well as the data files used in this module from the exercise file section, of the schools on Pluralsight. Now let's look into this data folder. Here you can see all files which we will import in this module. And here is the 0/1Sample. csu file which we will import in this clip. It is a very small CSV file. Let's open this file in a notepad application. As you can see this CSV file has two columns and four rows. Here we have marks as scored by four students in two classroom tests. One in physics and one in chemistry. Now let's close this file and move to RStudio to import this file in the R environment. So here we are in RStudio. Before importing the CSV file we have first set the working directory for the current R station. We have used the setwd command with the file. path method as discussed in the previous clip. We have set the working directory to the location where we have all our script files for this demo. Next on line number four we have created a variable file, which will contain the file path of the CSV file. And because we have placed all our data files in the data folder we have again used the file. path technique to create the path. So the first string is data because we have placed the CSV file in the data folder. And the second string is our file name itself. So here we are making use of relative path. Relative with respect to the current working directory. Now let's execute line number two and four. So our file path is set, now let's use this file path to import the CSV file. Here on line number five, we are passing the file variable to the read. csv method. And we have assigned the output of free. csv method to the my. data variable. Let's execute this line. So the CSV file is successfully imported and assigned to the variable my. data. Let's look a at its structure using str command as shown here on the line number six. Let's execute this line. So as you can see in the console, the my. data is a data frame with two variables and four rows. You can also print the contents of this variable using using my. data variable name as shown here on the line number seven. Let's execute this line. So here is our data frame on the console. So in this clip we discussed the use of read. csv method to import a CSV file. Now in the next clip we will learn a more generic method, read. table, which can be used to import tabular data. In fact, read. csv method internally uses the read the read. table method only. So let's learn more about read. table method in the next clip.

Import Table

Tabular data is arguably the most common format to store data. Tabular data is a generic structure which uses rows and columns to arrange and store the data. Let's look at few specific examples of tabular data. In the previous clip we looked at CSV file which is a special case of tabular data in which we use comma as a value separator and dot to represent decimal points.

And in another variation of CSV format the separator is semicolon and the comma is used for decimal points. Tab delimited format is another very popular format to store data, where values are separated by tab and dot is used for decimal points. Some tab delimited files use comma also for decimal points. So depending upon your requirement, you can set the separator and the character for decimal points. In R you can use the read. table method to read and import tabular file. Here's a typical uses of read. table method. You can pass the file path to the read. table method. This method will return a data frame. By default the header is set to false. Moreover, the default value of separator is whitespace. Which means the separator could be one or more spaces, tabs, new lines, or character. The default character used for decimal points is dot, so if your file has headers, or different separators, or different decimal point character, then you will have to save these parameters also. This read. table meter has several other parameters also. Let's look at few important parameters. You can set strings as factors to false, if you don't want R to treat character strings as factors. You can also explicitly provide class names for different columns of the tabular file using colclasses parameter. Other than that you can use the skip parameter to specify the number of lines to skip in the file before reading data. You can also set the value of nrows explicitly to set the number of rows to read from the file. Now let's take a demo to see the use of the this read. table function.

Demo: Import Table

In this clip, we will import this 02Sample. txt file. Let's open this file in a Notepad application. This file contains the same data which we saw in the data frames clip in the data structure module. I have simply placed those values in this file in a tab-delimited form. I have also included a comment on the first line, which we want to skip while reading this file. Moreover, we have headers in this file. The first column contains student names. We want this column to be character type. The second column contains gender. So this should be factor type. The third column contains weight. So this should be pneumatic type. Last to columns contain physics and chemistry subjects. And we want these two columns to be read as integer type. Now lets close this file and then import this file in R. So here we are in RStudio. First we have set the current working directory just like our previous clip in this module. Let's execute this line. So the working directory is set. Next on line number four, we are creating the file variable which will store the file path to the file to be imported. The file name is 02Sample. txt, placed in the data folder. Let's execute this line. So the file path is created. Next, from line number five to eight, we are using the read. table method to read the tab delimited file. So the first pattern we did is our file variable. Next, we have set the header parameter to TRUE because we have headers in our file. Then we

have set the skip parameter to 1 because we want to skip the first line. We can also set the call classes if you want explicitly mention the column types. We can use a character vector for this. So, we want first column to be character type, second column to be factor, third column to be numeric, and the fourth and fifth column to integer type. If you do not provide call classes, then R framework will determine the type of its own depending upon the content in that column. So if you are finding that R is not correctly setting the type the way you want, then you can set the call classes explicitly. Here we have assigned the output of read. table method to the variable my. data. Let's execute this statement. So the import process is completed and the result is stored in the my. data variable. Let's see the structure of my. data using the str function as shown here on line number nine. Let's execute this line. So as you can see in the console, we have a data frame with four observations and five variables. Moreover, we have desired types for each column. So in this clip, we saw how with the help of read method, we can read and import tabular data. However, so far in this module, we have imported the local files available on our local machine. But sometimes the data is available on the remote machine, and you may have to bring that file, to your local system and then import that data in order to perform data analysis. So now in the next tape I'll be talking about, importing data from a URL.

Import from URL

In many data analysis projects, the data file may not be available on your local machine beforehand. And you may have Downloaded from a URL, or a Universal Resource Locator. Well R has an in-built web server, which can be used to Download files from a URL. For example is the file is a table file, then in the redirect method you can parse the URL of the file itself and this will download and encode the file. However if you will use this line, then the file will be downloaded to a temporary location. So sometimes, it will be better if you can download the file, to your working folder. In R, you can use the download. file method to do so. The first parameter is the URL of the file to be downloaded, and the second parameter is the local file. So the file available in the URL will be downloaded, on your local machine as local file, and then you can use any method to read that local file. So if that file is a tabular file, then you can use the read table function to read that file. Now let's take a demo to see this in action.

Demo: Import from URL

Here we are in R Studio, here on line number one we have a URL, to one of the most popular machine learning dataset, that is Iris dataset. This dataset is available on UCI Machine Learning

repository site, and here is the URL, to download this dataset. We have assigned this URL of the iris dataset to the variable URL on the left hand side, of the assignment operator. Next, on line number two, we have first set our working directory just like our previous demos. Then on line number 3, we are creating the path to local destination file, where the downloaded file will be placed. Here we want to place the downloaded file in the data folder, with a name of 03downloadedfile. data. On line number 4, we are downloading the file using download. file method. The first parameter is our source URL. While the second parameter is the local destination file. And once it will be downloaded, then we can use the read. table method to read the local file. Because this dataset has comma as the separator, we have set the separator parameter to comma. Now let's execute lines from 1 to 5. So as you can see on the console, the file has been successfully downloaded, and saved to the data folder on the local system. And then using the read. table method we have imported the local file, to create the data frame my. data. So now we can look at the structure of this data frame my. data using STR method, as shown here on the line number six, let's execute this line. So this data set has 150 observations, with five variables. So you can use this technique if data is available on some URL, and you want that downloading process also to be part of the script. However this technique is used for very simple scenarios. But if you have some more complicated and advanced scenarios, such as downloading from a secure site, or handling redirection, or password authentication. Then you can look at R Curl package, this will help you in most of the web's crafting activities. So now you have learned to import tabular files, from local and remote machines. Let's learn to import a couple of other common file formats, starting with XML files in the next clip.

Import XML Files

XML, or extensible markup language, is another very common and open source file format to store data. Here's the typical XML data, this XML data contains information about two students of a classroom. For each student we have name and marks scored in two subjects, physics and chemistry. Now let's see how to input XML data in R.

Demo: Import XML Files

In this clip, we will import this 04 sample. XML file, containing the XML data. Let's open this file in a Notepad application. So here you can see we have details of four students, for each student we have name, as well as physics and chemistry marks. Now let's close this XML file and import this file in R. So just like our previous clips, we have first set our working directory, and then the file

path. So we'll be importing this 04sample. xml file, placed in the data folder. Well in R there are several packages which can help you to import XML data. However in this clip, we'll be using the XML package, so first on line number five we are installing the XML package using install. packages function. Then on line number six, we are loading the XML package using the library function. Next from line number seven to nine, we are using the XML to data frame function available in XML package, to import the XML file in a R data frame object. There are other functions also in the XML package, to read the XML data in different fashion. So in this XML to data frame function, the first parameter is the file object, which refers us to the local file. Instead of the local file, if you have some URL also, then you can pass that URL. We already seen a technique to get data from remote places in the previous clip, then as a second parameter we have explicitly specified the column classes, by setting the call classes parameter. If you will not specify this parameter, then R framework will decide of its own, based on the data. We have also set the strings as factors to falls, so that strings should not be considered as factors. So now let's execute lines, from one to nine to import the local XML data. So the import process is completed. Now let's see the structure of the imported data frame object, using the STR function as shown here on the line number 10, let's execute this line. So as you can see, we have four observations and three variables. The data types are also properly set. So in this clip, we learn to import XML data. Now, in the next clip, we will learn to import another very popular file format, that is, Excel file.

Import Excel Files

Excel files are also very popular, to store data. In R, there are several add on packages, to import Excel files in R. However, in this clip, we will take one such package, known as XL Connect. This package will work on all major operating systems such as Windows, Mac and Linux provided you have a Java installed on your machine. This package can be used to read both xls as well as xlsx file. So if this package is not installed on your machine, then you can install this package using install. packages function. And once the package is installed, we can first load the package using the library function, then you can read the actual sheet using the readWorksheetFromFile method, you can pass the file path as the first parameter. And then you can either provide the sheet number, or you can provide the sheet name. Now let's take an example to see this in action.

Demo: Import Excel Files

In this clip we'll import data from this XL file, 05sample. xlsx, let's open this file. So as you can see we have exactly the same content here, to that of the previous demo. Just this time the file is saved as xlsx file, let's close this file and import it in R. So here we are in our studio, and just like previous clips of this module, we first set to working data to using W command. Then we are creating the file variable using file. path method, and we have passed the filename 05 Sample. xlsx. Next, we are installing the XLConnect package using install. packages method as shown here on the line number five. Then, on line number 6, we are loading the XL Connect package. And then from line number 7 to 9, we are importing the Excel worksheet using the readWorksheetFrom File method. The first parameter is the file variable. Then we have used one as the value of the sheet parameter. Means we want to read the first sheet of the Excel file. We have also set the start row parameter, this parameter signifies that from which row, you want to start reading. And because we have a comment on the first line, we have set the value of the start row parameter to two. We are assigning the output of this function to the my. data variable. So now let's execute lines from 1 to 9, to install and load XL connect package and then import XL file. After importing the XL file, if you want to see the structure of the my. data, then we can use the STR function as shown here on the line number 10, let's execute this line. So, as you can see, we have a data frame with four observations, and five variables. However, if you look at this structure, we have character type for column one and column two, and for rest of three columns, the type is numeric. Well, XLConnect does not support factor or integer types as did. So if, you want to do such transformations, then you can use a very useful function that is transform function. You can use this function to transform the existing object, as per your requirement. So here, from line number 11 to 14 we are using the transform method. The first parameter is our data frame, my. data. Then we have used explicit coercion methods, to change the type of columns. We have used the as. factor method to covert the gender column, then we have used as. integer methods to convert last two columns, to integer type. Now let's execute the statement, so the transformation of the data frame is completed. Now let's see the structure of the transformed data frame object, using STR meta dash on here under line number 15, let's execute this line. And as you can see in the console, the gender column has been changed to factor. And the last two columns have been changed to a teacher type. So in this clip, we learned the use of Excel connect package, to import Excel files into our R session. There are other packages also you can try to import Excel data in R. Well, so far in this module, we have learned to import some very popular file formats, such as. csv file, generally tabular file,. XML file and Excel file. However you can use R, to import data from other commercial data analysis softwares also. So let's briefly look at the process to import data, from these proprietary formats in the next clip.

Import Other File Types

You can also import datafrom other proprietary formats also, such as. SPSS, Stata, Minitab and SAS. And the easiest way to do so, is to use the foreign package to importa data from these file types. So if you have some data at a level in these software packages, then you can bring that data in the R environment. And then can use the power of our framework, to analyze that detail. I will not go into the details of the important process for each format, but lets see the typical process you can adopt to import these proprite the data formats in our. So first we can lure the foreign package using the, then to read different types of formats, there are different functions available in foreign package. So you can use the read. spss function to read the SPSS data file, you can use the read. dta function to read Stata binary file, read. mtp function to read the Minitab portable worksheet. And read your export function, read the SAS XPort format library. Foreign package supports other five formats also. So, if you want to learn more about the supported file formats, then you can check the help page for foreign package, using the Help command. Well, so far in this clip, we learn to import local and remote flies in a wide variety of formats. But sometimes you may need some quick data to test your logic, or to learn new concepts. So in those scenarios, built in datasets, can come in handy. So, in the next clip, we'll talk about importing built in datasets.

Import Built-In Datasets

Many R packages also contain these datasets, which may be useful to you. These datasets can be used if you are learning new concepts, or trying and testing new things, and you need some data to work on. In those situations, such built-in datasets can be very handy, because you do not have to search for data. Moreover, in most of the cases, these are public datasets, therefore, you can use them in your research activities also. But you should always check the details, before using these datasets for publications. So to import built in data sets in R, you can first load the required package. And if the package is not installed, then you will have to install that package first. After loading the package, you can use the data command. Inside the data command, you can pass the data set name, to be imported. Then you can use this dataset name as any other variable, and you can access its contents. Now, let's take a short demo to see the process.

Demo: Import Built-In Datasets

Here we are in R studio. In this demo, we're using the datasets package, which is installed with the base R version itself. This package contains various datasets which you can use. Let's load that

package using the library function as shown here on the line number one. Let's execute this line. So the datasets package is loaded. Now, if you want to see the available data sets in this package, then in the data command, you can pass the package name through the package parameter as shown here on the line number two. Let's execute this line. And as you can see, a separate tab is opened in R studio. And here you will find different datasets available in this package. We have the Iris data set also in this list. Now let's import this Iris dataset to the current R session. So, to import the Iris data set, we can simply put the Iris data set name in the data command as shown here on the line number three. Let's execute this line. >> So the Iris data set is successfully imported to the current data session. And now you can use iris as just like your own variable and you can work on it. So if you want to see the structure of this iris dataset, then you can use the str command as shown here on the line number four. Let's execute this line. So as you can see, we have 150 observations with five variables in the iris dataset. So in this clip, we'll learn the use of data command to import built in datasets to the current R session. Well, so far in this module, I have talked about importing data from variety of files and built in datasets. But in many real world scenarios, you would like to extract data from your existing databases, and then work on them in the R environment. Therefore in the next clip, I will list out a few popular R packages which can help you in this process.

Import from Database

Various companies store their data in the form of databases. So if you want to perform data analysis on that data, then you need to import or extract data from those databases to the R environment. R has a very good support for major databases. There are various add-on packages in R which can help you to connect and import data from relational databases as well as NoSQL databases. RODBC and DBI are two very popular packages which are used to import data from various relational databases. RODBC package can be used to import data from various databases that support ODBC connectivity such as Microsoft SQL Server or Microsoft Access or MySQL and many more. DBI of database interface package is another very powerful package which can be used to import data from many popular databases. DBI package provides unified syntax for importing data from a range of database systems. There are database specific packages also in R. So you can use the RMySQL package if you are working with MySQL database. You can use the ROracle package for Oracle database. RPostgreSQL package for PostgreSQL, and RSQLite package for importing data from SQLite database. So as you can see, R has a very wide support for various types of relational database systems. Along with relational databases, R can be connected to various NoSQL databases also. You can use RMongo or rmongodb package to

import data from MongoDB. RCassandra package to import data from Cassandra. And R4CouchDB package to import data from CouchDB. rhbase package that is a part of RHadoop, can be used to input data from HBase, which is a part of Hadoop ecosystem. Well, I have listed out several packages to collect and import data from various database systems. So depending upon your requirement, you can further delve into the recquired packages. You can also take help of available vignette documentation and other online materials. Now let's take a quick demo where I will show you the process of importing data from a MySQL database using RODBC package. Similar approach can be adopted to input data from other databases also which support ODBC connectivity.

Demo: Import Database Using RODBC Package

To import data from any database using RODBC Package, you need to first setup and configure the ODBC DSN or Data Source Name. DSN is the name of the ODBC connection to the database from which you want to import data. In this clip, we will import data from a MySQL database. Well, configuring ODBC DSN is beyond the scope of this course, but here are a few links that can help you to configure DSN on your respective operating systems. These links are for MySQL database, but steps are very similar if you want to connect to other databases that support ODBC activity. Now let's look at the R commands to connect and import data from the MySQL database. So here we are in R studio. Well, to use the RODBC package, you need to first install this package. So on line number 1, we're first installing the RODBC package using the install. packages command. Let's execute this line. So the RODBC package has been installed. Now, let's load this package using the library command action here in the line number 2. Let's execute this line. So this RODBC package has been loaded. Next, on line number 3 we're creating an RODBC connection object connect, using the RODBC connect command. In this command, we have passed the DSN name. I have already created a DSN named as MySQL connection to connect to MySQL database on my local machine. So once the connection is set up, then you can use this connection to make SQL queries using sqlQuery command to bring data in the R environment. So here on line number 4, we are using a very simple select query to get all data from classroom table of test database. We have stored the output of the SQL query to my. data object. So now let's execute line number 3 and 4 to set up the connection and then to import data from the test database. So the data has been imported. Now let's look at the imported data using the variable name my. data as shown here on line number 5. Let's execute this line. And here we have the data on our console which was imported from a MySQL database. Well, here in this clip we have

imported data from MySQL database, but you can import data from other databases also in a similar fashion.

Summary

So, we have reached to the end of this module. In this module, we learn to import data from a variety of sources. We started with various types of local and remote files. We first learned the use of read. csv method to import data from a CSV file. And in general, you can use the read. table method to import data from tabular files. Then we learned the use of XML package to import XML files and XLConnect package to import XL files. We also discussed about foreign package to import data from other data analysis softwares. Then the next section, we'll learn to import data from built-in datasets. And we'll learn the use of data function to load the desired data set into the current R session. And finally, in the last section, I listed out various packages to import data from various databases. R has support for relational as well as NoSQL databases. We also took a demo to see the use of RODBC package to input data from MySQL database. Well, this was all about importing data in R. Techniques learned in this module will help you to get the required data easily and efficiently. So now we have the data to work on. So in the next clip, I'll talk about some basic data analysis tasks to explore the available data. This will help you to get some valuable insights in the available data. So in the next module, you will learn to work with data, and that will create a solid foundation for your data analysis endeavors. So, see you in the next module.

Exploring Data With R

Introduction

Hi, this is Abhishek Kumar, and welcome to the 10th module on R programming fundamentals, which is Exploring Data With R. Well, so far in this course, we have difficult radius aspects of code R programming. Then in the previous module, we learned to import data from a variety of sources. Now, in this module, we will apply the learning of previous modules to explore, and extract knowledge from a given dataset. So, in this module, you will learn to answer questions like, given a dataset, what can you see about that dataset in a broad sense. So we will discuss various key statistical indicators, which can help you to summarize a dataset. We will also discuss the user base R functions, to perform such kind of analysis. This module is also filled with demos, just like

previous modules. In this module, we will work on a very popular structure dataset. However sometimes you may have to spend some time to prepare your data, before performing any analysis. Concepts learned in the previous modules of this course, will definitely help you in such pre-processing activities. so once you have a well formatted and a processed dataset, you can follow the concepts discussed in this module, to explore your data set. So by the end of this module, you will not only learn about various statistical indicators and their significances, but also learn to use them in our framework, to explore a given data set.

Outline

Here's the module outline, we will first learn to explore the overall structure of the dataset, such as number of rows, number of columns, and data types for each column. Then we will learn to analyze continuous or quantitative data. We will mainly talk about central tendency, and the spread of data. After analyzing continuous data, we will turn our attention towards categorical or qualitative data. Where we will learn to find some key information, from the categorical using various odd functions. But before getting into the details of data exploration, let's first discuss what are the typical types of data you will analyze, or explore in a real world scenario.

Types of Data

In data analysis projects, you will mostly analyze two types of data. First is the categorical data. And second is the continuous data. Let's look at the difference between these two types. So, categorical data, as the name suggests, are used to represent categories. Suppose you want to represent colors of different cars, or human gender. Then in such cases the values can belong to only certain categories. Means the color of the car can be red, or blue, or green, or so on and the gender can be male or female. These categories are non overlapping in nature, so a person can be either male or a female. Move over is such scenarios the options, or the number of categories, are also limited. So we put such type of data in the categorical data category, and such variables are know as categorical variables, or categorical features. And in R we typically use factors to represent such data. And the levels of the factor are used, to represent different categories. We have already discussed factors in the data structure module. So if you want to know more about factors, then you can go through the data factors clip in the data structure module, so this was about categorical data. On the other hand, for continuous data a variable can take a value in virtually unlimited number of options. Take the example of car mileage. So the mileage can be anything like 21. 1 or 24. 3 or so on. Or take the example of Bosun's height, or weight, or even age.

In these scenarios, the number of options for a variable is almost infinite. So we call these types of data as continuous data, and such variables are known as continuous variables, or continuous features. And in R, we normally use numeric or integer data types, to represent such data. Sometimes, you can put a variable in either categorical, or continuous category. For example, one might argue that if you are taking age, especially in integer, then there are only limited options. Well in such cases it's totally your call, how you want to treat a variable, and the choice is largely dependent upon the type of analysis you want to perform. So now you must be having a clear understanding, of continuous and categorical data. Now we will move further, and start learning to explore data. In the next clip, we will take our first step toward data exploration, by getting an overall structure of a given data set.

Overall Structure

To get the overall structure, is the first step towards exploring the given dataset. Where if you would like to know, how many observations or instances we have in the data set. Or how many features are there? And what are their data types? We would also like to see the sample data set, probably if only but this will give you an overall idea, about the data set. Well to work on various aspects of data exploration, we will take a public dataset in this module.

Example Dataset

We'll be working on Iris dataset, which we had talked about in the previous module also. Iris dataset is very popular dataset among data analysts. This dataset contains 50 samples, from each of the three species of the iris flower. The tree species are Iris-setosa, Iris-virginca, and Iris-versicolor. For each sample a value of four features has been measured. The four features are, length and width of petals and the sepals of the flower. Well in R, this dataset is available in the datasets package. Which comes with a standard installation of R. So we will import an iris dataset, from the datasets package. Now let's take a demo, in which we will first import a dataset. And then we will try to get the overall structure, of the dataset.

Demo: Overall Structure

Here we are in our studio, on line number 1, we are first loading the data sets package using the library command, let's execute this line, so the data sets package is loaded. Next on line number 2, we are importing the iris dataset to the iris variable. Using the data command as discussed in the previous module. Now let's execute this line, so, the iris data set has been imported. Next on line

number three, we are using the STR function to oversee the overall structure of this dataset. We are passing iris in the STR function, let's execute this line. Now, let's examine the result on the console. So, this dataset has 150 observations, with five variables. First four features are numeric in nature, and contain Sepal Length, Sepal Width, Petal Length and Petal Width, for each flower sample. So as you can see, these four variables are continuous variables. The fifth feature is Species, which contains the type of species each sample belongs to. So this is a categorical variable, because we have non-overlapping categories of species. And here, the data type is Factor, and there are three levels of the factor belonging to three types of species, setosa, virginica and versicolor. So the str command will give you an overall structure of the dataset. If you want to look at the sample of this dataset, then you can use the head command, on line number four we are using the head command, and we are paused iris dataset, to the head command as parameter. Head command returns top few rules of the dataset, lets execute this link. And then here on the console, we have a sample subset of the whole iris dataset. So the top few rows are displayed here, head is a very handy function, if you just want to see how the dataset looks. So in this clip, we saw how, with the help of STR, and head function you can look at the overall structure of the dataset. Now we will go into the details of this dataset, and we'll analyze various continuous and categorical variables, starting with the continuous variables in the next clip.

Analysis of Continuous Data

For analyzing continuous data, we will mainly focus one two aspects. First is The central tendency, which will give you a general idea about a dataset, rather than individual values. And the second is spread or dispersion of data. There are various statistical indicators or techniques, which can help you to understand both aspects. So let's start with the central tendency in the next clip.

Central Tendency (Mean)

As the name suggests, measure of central tendency are used to find values in the middle of the data. And one of the most common measures of central tendency is the mean, also known as arithmetic mean or average, let's learn more about mean using an example. Suppose this is a classroom of eight students, and they scored these marks in a classroom test. So to calculate the mean or average of marks, we can simply get the sum of all values, and then divide it by the total number of values. So here, we can add all marks, then we can divide by the number of students,

this will come out to be 75. So this single figure, will give you a general idea how the students have performed in the test. So this will give you an idea about the center of data. So if the mean value is on the higher side, then we can say that there are students who have scored well. But if the mean value is on the low side, then we can say that there are students who have scored less, obviously the mean value will not give you a concrete picture, but it is a very handy value to get a general sense of how the students have performed. Now let's learn to get mean value, using odd in a demo where we will work on our average data set

Demo: Central Tendency (Mean)

Here we are in R studio, on line number 1, we have first loaded the data sets package, using library command, and then on line number 2 we imported the iris dataset using data command. Next on line number 3, we are using the mean function to calculate the mean value. If we want to get the mean value of sepal length column data, therefor we have used Sepal. Length after the dollar sign. This will return the corresponding column data, from the data frame. We have already discussed such subletting techniques, in the data frame clip in the data structures module. Now let's execute lines from one to three in one go. So first the dataset's package was loaded, then we have imported the iris data set in a data frame. And finally we got the mean value of sepal. length. So in this manner you can get average or mean value, of other continuous variables as well. Well, mean is not the only measure of central tendency. Another such measure is median, which we will take in the next clip.

Central Tendency (Median)

Median is another very useful major of central tendency. The median represents the value at central position, if you have an ordered list of values. Let's take our classroom example once again, to do more about median. So to calculate median for this data, let's first arrange these values. Here we have arranged the values, in an ascending order. Next find the value at the halfway mark. Well, because we have even number of values in the set here, therefore there are two values at the center. So in this case we have to take the average of these two values to get the median value. So in this example the median will be the average of 75 and 80, and that is 77. 5. So the median, will tell you about the value at the center of a set of values. Now let's see, how we can find median for a continuous feature in our.

Demo: Central Tendency (Median)

Here we are in RStudio, again, in the first couple of lines, we are learning the datasets package and then importing the irish dataset using data command. Next on line number three, we have using the median function to get the median value. Here we are trying to extract the medial value, of sepal length column data in the iris dataset, let's execute these three lines in one goal, and we get the required median value. In a similar fashion you can calculate the median value, for other continuous features also. Well so far we have discussed two measures of central tendency, mean and median. But these two values only, cannot give you a clear picture about the data, as these values are easily affected by outliers in a given data. So in the next clip, we will see why mean and median values alone, are not sufficient to describe the data.

Central Tendency: Why Not Sufficient?

Mean and median values are not sufficient, to say something concrete about data. Let's look at one example to see why it is so. Here is the classroom data once again, which we saw in the previous clips. So for this data the mean value is 75, while the median value is 77. 5. But let's look at this, another classroom scenario, where again we have months of eight students in a classroom. But if you will calculate the mean and median for this dataset, then you will get the same mean and median values. Why did so? Well, because mean and median values, only provide the central tendency, they do not tell you about individual values. If you look at this first set of values, here you will find all of the marks are greater than or equal to 60, with the minimum at 60 and maximum at 90. While in the second set we even have a value of 25, and we have two values on the very higher side with two scores of 95 and one score of hundred. So as you can see even the data set are quite different in nature, but then also we have the same value of mean and median. That's why, to describe a set of values, you should also consider the spread or dispersion of data, other than central tendency. So in the next few clips, I will talk about few measures of the spread, starting with the range in the next clip.

Spread (Range)

Range is used to describe the spread of data. Let's take the classroom example once again to know more about range. So the range is the difference of maximum and minimum value in a set of values. So in this case, the maximum value is 90 and the minimum value is 60. So the range is 90 minus 60 that is 30. Now let's look at these two sets of marks with same Mean and Median values, which we had discussed previously. So, for the first set of values, the range is 90 minus 60, that is 30. While for the second set of values, the range is 100 minus 25 that is 75. So in the

second set of values, we have a wider range of values. So as you can see with the help of range value, you are in a better position to describe the data. Now let's take a quick demo to see how we can calculate the range in R.

Demo: Spread (Range)

Here we are in our studio, and just like previous clips of this module, we first load the datasets package and then import the iris dataset. Next on line number three, we are using the range function. And we want to find out the range of simple length column data in the iris dataset. Let's execute lines from one to three, and as you can see on the console, the range function returns two values. First is the minimum value, and the second is the maximum value in the Ssepal length column data. If you are interested in finding the difference between these two values, then you can use the diff function as shown here in the line number four. Here we have passed the output of range function to the diff function. So this will return the difference between maximum and minimum value. Let's execute this lane. And here we have the result of 3. 6, which is the difference between maximum and minimum value, for simple length column data in the Irish data set. So as you can see, the range is a very useful measure of spread. But it has its own limitation. Range highlights only the minimum and the maximum value in the data. It does not provide any information about spread of intermediate values. So in the next clip, we will discuss another very useful measure of dispersion that is quartile.

Spread (Quartiles)

As the name suggests, quartiles divide the data into four parts, where each part contains the same number of values. Let's look at one example to understand quartiles. Let's take our classroom example once again. So suppose we want to find quartiles for this set A, then first we can arrange these marks in an ascending order. Then we will try to find the median. Here in this data set because we have even number of values. There are two middle values, and therefore the median will be the average of these two values. So the median is 77. 5. This median is also termed as Q2 or the second quartile. Next we can divide the set into two parts. In the first part we will have values lower than Q2, and the second part values are greater than Q2. Then again we will find median of lower numbers and higher numbers. So the median of lower numbers will be calculated in a similar fashion, and here's the result. This median is known as the Q1, or the first quartile. Similarly, the median of higher numbers can be calculated, and this median is known as Q3 or third quartile. Well, if you will try to calculate values of Q1, Q2 and Q3 using R function, then

you might get a different result. Because in R, there're different types in which quartiles are computed. However, for a large data set, the values of Q1, Q2, and Q3 will be nearly same in all types. Anyways, the point I want to emphasize here is that the job of these quartiles, Q1, Q2, and Q3, is to divide data into four equal parts. So the values of the set are arranged in an ascending order. Then these quartile values will divide the set of values in four parts, each containing 25% of values. So from minimum value to Q1, there will be 25% of values, from Q1 to Q2, next 25%. From Q2 to Q3, next 25%. And finally, Q3 to maximum. The final 25% values. So as you can see, these five values can help you to understand the spread of data. In fact, these five values have a very special name. That is five point summary. Now let's look at a demo to see how we can find five point summary in R.

Demo: Spread (Quartiles)

Here, we are in our studio. In the first couple of lines we are first loading the data sets package, and then importing the Iris data set using data command. Next on line number three we are using the summary function, to get five point summary. Here we have passed the simple length column data to the summary function. Let's execute lines from one to three, and here is the result on the console. So we have minimum value, first quartile, medium or second quartile, third quartile, and the maximum value. The result also includes the mean value. So between the minimum value of 4. 3 and first quartile 5. 1, there will be 25% values. Similarly between first quartile 5. 1 and second quartile 5. 8, there will be next 25% values. Then, the next 25% values will be between 5. 8 and the third quartile 6. 4. And the final 25% values will be between third quartile 6. 4, and the maximum value 7. 9. So, you can use the summary function to get these summary statistics easily. However on line number three, we have applied the summary function on the sepal. length column only. But you can use the summary function on the whole data set in one go also. Here on line number four, we have passed the iris data set itself to the summary function. Let's execute this line. And on the console, we have the summary statistics for all continuous columns. For categorical columns such as species, the summary function returns the total number of rows in each category. We will talk about categorical data later in this module. However, as you can see, some of these are very useful R function that can be used to know the spread of data. Moreover, there is a feasible way to represent this five point summary. This is known as Box Plot, which we will discuss in the next clip.

Spread (Box Plot)

Box Plot, also know as Box and Whisker Plot, is a great way to visualize quartiles and in turn the spread of data. Let's take our classroom example once again, to understand Box Plots. Here are five point summary values for this data set. So we have minimum, quartile one, quartile two, quartile three, and maximum. And box plot is basically the visual representation of these five values. Let's see how box plots are created. Let's draw a number line. Then we will take the minimum value and place it on the number line. This minimum value will be our whisker. We extend this whisker till the first quartile, and from the first quartile, we will start our box. And this box will be extended till the third quartile. Then again we will have a whisker, and which will be extended in the maximum value. And finally, the second quartile, or the median of the whole data, will be placed in the box as a bold line. So this is how we can create Box Plot. So as you can see just by looking at this plot, we can have an idea of the spread. This box contains 50% of data between quartile one and quartile three. And 50% of data will be outside this box. Now let's compare the Box Plot for these two sets; Set A and Set B. Here are the five points from the results for the Set A. In the similar fashion, you can calculate the five points summary information for the set B. Now let's plot the box plot for both the sets on a single number line. So here's the box plot for the set A, and here's the box plot for the set B. So just by comparing these two box plots, you can find that the set B has a larger range. Moreover in the Set A, 50% of values are in this box, which is smaller in size than in the case of the set B. You can also these other things, but the crux of the box plot is that it is a quick way to analyze the spread of data. Well in this example, we looked at the horizontal box plot. But many times, you may find vertical box plots also. However, vertical box plots are same as horizontal box plots, only the presentation is different in both cases. Now let's look at various components of this vertical box plot. Starting from the bottom, we have the minimum value. Then we have the quartile one, which is also referred as the lower quartile. So 25% of data will be below this lower quartile. Next we have the second quartile or the median. And 50% of data will be below this second quartile. And 50% of data will be higher than this second quartile. Next we have third quartile, which is also known as the upper quartile. So 75% of data will be below this upper quartile, and 25% of data will be above this upper quartile. And finally, we have our maximum value. Now let's include one more piece of information in this box plot. To do this, we will first restate the definition of maximum and minimum. So actually, maximum and minimum values are determined after excluding outliers. So what are outliers? So, a value is considered as an outlier when that value is significantly distant from other values in the data. Typically in any data analysis, outliers are first removed, before drawing inferences from data. So sometimes you may find these dots at one or both ends of the box plot. These dots represent outliers. So while upper dots represent values that are higher than 1. 5 times the third quartile, the lower dots on the other hand represent values lower than 1. 5

times the first quartile. So now you have understood various components of a Box Plot. Let's learn to create a Box Plot in R.

Demo: Spread (Box Plot)

Here we are in our studio, and just like our previous clips in this module, we first load the datasets package, and then input the iris dataset using data command. Next on line number three, we are creating a box plot using box plot function in R. In the box plot function, we have passed the Ssepal. length column data. Let's execute lines from one to three. So as you can see we have the box plot in the plot tab on the right hand side. By default, the box plot will be drawn vertically in R, but if you want the horizontal box plot, then you can set the horizontal parameter to true action here on the line number four. Let's execute this line. And here we have a horizontal box plot. So you can use the box plot to analyze the spread of data. If you want to know the stats for this box plot, then you can use the boxplot. stats function, as shown here on the line number five, where we have passed the Ssepal. length column data. Let's execute this line. This function returns a list with different elements. The stat element contains the five important values used to create the box plot. First is minimum, then we have first, second and third quartile, and finally we have the maximum value. This list also has this out element, which will include values if the data has outliers. So the outliers will be the values beyond the viscus. Well so far we have plotted a single plot for one column, but you can have multiple box plots as well. Here on line number six, we are using the Box Plot function to plot the first full columns, containing continuous data. Let's execute this line. And here we have our result. We have four box plots. So as you can see, the second box plot corresponding to the Ssepal width has few dots also, means this column has some outliers. Well this was all about box plots. So with the help of box plot, we can divide a data set into four bins, where each bin has 25% of values. But bin size or the range of values in the bin is determined by data. Now in the next clip, we will talk about another reasonable technique to represent the spread of data, that is histogram. However, unlike Box Plots, in the histogram, the bin size, or the range of values in the bin will be fixed. But the number of values in each bin can be different. So let's look at the histogram next.

Spread (Histogram)

Histogram is also a very popular way to visualize and analyze the spread of data. Let's understand histograms again using the classroom example. So suppose we have to divide the marks obtained by these eight students into four bins like this. First bin will contain values in the range of 51 to 61.

The second bin will contain values in the range of 61 to 71. The third in the 71 to 81, and fourth in the range of 81 to 91. So the range of each bin is ten. On the right hand side of this table we have the number of values or count in that bin. So the first bin contains two values, the second bin contains one value, the third value contains three values, and the fourth bin contains two values. So we can use this table to create a histogram, so the histogram is visual representation of this table only. Here's the histogram for this dataset with every mentioned been ranged. So by looking into the histogram, you can visualize the spread of data. Now let's look at one demo to create histograms in R.

Demo: Spread (Histogram)

Here, we are in R Studio, and in the first couple of lines, we are loading the datasets package and then importing the iris dataset. Next on line number three we are using the hist function to create a histogram, using the sepal length column data of iris data set. Let's execute lines from 1 to 3, so by looking into this histogram, we can analyze that most of the values are in the middle and we have fewer values on the extremes. You can also customize this histogram. Here you can see the title of the histogram as well as the X access level, are not looking appropriate. So you can change them, here in the statement from line number 4 to 6, we have set the values of two extra parameters. So we can set the main parameter to set the title of the plot. And we can set the xlab parameter to set the x-axis title. So let's execute this statement. And we have the desired histogram with custom plot title and axis title. So, in this clip, we learned the use of hist function to create a histogram and visualize the spread of data. Now in the next clip, we will look at two more statistical indicators which can help us to understand the spread of data, data variance, and standard deviation.

Spread (Variance and Standard Deviation)

Variance and standard deviations are very common and useful statistical indicators, to get an insight about spread of data. Let's learn more about these indicators, using our classroom example. To learn more about the variance and standard deviation, let's see how these are calculated. So here are our values in the data. Next, we compute the mean value for this data, then we find the difference between each value and the mean value. So the first value is 60 minus 75 equals to minus 15, similarly we can calculate that their values. So next, we find the square of these values, so here is the squared result, and then we find the sum of all these values, and here's the result. Next we divide the sum by the number of values in the set, and because we have 8

values, we are dividing the sum by 8, so this result is the variance of data. So variance for this set of values is 106. 25. Next, to calculate the standard deviation, we can simply take the square root of this value, so the standard deviation is 10. 30. Well, don't bother at all if you're not able to remember the steps of calculating variance or standard deviation. I have shown this process of computing these values, because this will help you to understand what exactly going on here. So, if you look closely, here we are trying to find how far each value in the set is from the mean value. And because we do not care whether it is a negative or a positive, that's why we are taking the square of the distance in the next step. So ultimately we are trying to find out how far individual values are from the mean. So variance and standard deviation will help you to analyze the spread of data with respect to mean or the central value. The higher the values of these statistical indicators, more will be the spread. On the side note, I want to tell you that, if you calculate variance or standard deviation of this set of values using R, then you will find a different answer, because in R, instead of dividing the sum by total number of values, the sum is divided by one less standard to the number of values. Means in R, the sum will be divided by 7 instead of 8. However, for large data sets, it doesn't make much difference. Well, you don't want to go into such minute details, but the point I want to make here is that the variance and the standard deviation will give you a good idea about the spread of data. Let's compare the standard deviation of these two sets. So both sets have same mean and median values. Now let's see the standard deviation and variance for both these sets. So for the first set the standard deviation is around 10. 3 and the variance is 106. 25, while for the second set the standard deviation is approximately 20. 9 and the variance is 437. 5. So as you can see even though both these sets have the same central tendency, but the spread is different in both cases. The set B has more spread because the standard deviation and the variance of the second set it higher than the set A. So now you have a good sense of variance and the standard deviation. Let's learn to find these values in R.

Demo: Spread (Variance and Standard Deviation)

Here we are in our studio, and first we have loaded the data sets package, and then we have imported the H data set. Next on line number 3, we are using the var function to calculate the variance of sepal length column data of Iris dataset. To calculate the standard deviation, we are using the sd function as shown here on the line number 4. Here, again, we are calculating the standard deviation of sepal length column data. So let's execute all these full lines to get variance and standard deviation, and we have got the desert on the console. So the variance of sepal length column data is approximately 0. 69, while the standard deviation is approximately 0. 83.

So in this clip, we saw how with the help of var and sd function, we can evaluate variance and standard deviation of data. These indicators are very helpful in assessing the spread of data. Well so far in this module we talked about continuous data, now from the next clip, we will talk about another category of data, that is categorical data.

Analysis of Categorical Data

As mentioned earlier in this module, categorical detail is the type of data, where the value can be from a limited number of options only. Such as the color of the car or the human gender. Now let's look at few techniques to analyze the categorical data. Well, to explore and analyze categorical data, we will mainly look into two aspects. First, is the frequency distribution, and the second is the category statistics. So let's start with the frequency distribution in the next clip.

Frequency Distribution

Frequency distribution is typically the first step in analyzing categorical data. Let's understand this using our classroom example. Suppose in the classroom, we have the information about gender of each student. Now let's see the frequency distribution. So there are three students in the male category and five students in the female category. There is a visual representation of this frequency distribution also, known as the bar plot. Here's the bar plot for every mentioned frequency distribution. So just by looking into this bar plot you can see that there are more female students in the classroom than male students. Other than count you can also find relative proportion for each category. So 3 out of 8 students are male so their relative proportion is 0. 375 or 37. 5%. Similarly5 out of 8 students are female. So the relative proportion of female students is 0. 625 or 62. 5%. Now let's see how we can get frequency distribution in R.

Demo: Frequency Distribution

Here we are in R Studio, and just like the previous demos of this module, we are first loading the data sets package, and then we are importing the Iris data set using data command. Next on line number 3, we are using the table function to get the frequency distribution. We have passed the categorical data, and we're leveling the species column of Iris data set. Now, let's execute lines from 1 to 3, and we have the result on the console. So in the Iris data set, each of three categories has 50 number of samples. Next on line number 4, we are creating a bar plot, by passing the output of the table function, to the bar plot function. Let's execute this line. And here we have the bar plot on the right hand side in the plot tab. So, as you can see we have categories on the X

axis, and number of samples on Y axis. In the Iris data set, we have the same number of samples in each category. Next, on line number 5, we are calculating the relative proportion of each category by passing the output of table function to prop. table function. Let's execute this line. So we have 0. 333 in each case. If you want to see this result in terms of percentage, then you can simply multiply the output of prop. table by a hundred, as shown here on the line number 6, let's execute this line. And here we have our result with approximately 33. 3% values in each category. So in this clip, we learn the use of table function, to find the frequency distribution for categorical data. Now in the next clip, we will talk about category statistics, to find values of various statistical indicators by category.

Category Statistics

Other than frequency distribution, category statistics, or per category statistics can be a very useful technique to explore the dataset containing categorical data. Let's take the classroom example once again. Suppose we have both test marks, as well as gender information for all of the eight students in the classroom. So, the marks scored by the male students are these, while these are the marks scored by female students. So, it would be interesting to know the value of various statistical indicators category wise. Let's take the example of mean, or average value. So the average marks as scored by male students is approximately 73. 3. While average or mean marks scored by females students is 76. So by looking into these two average values you can see that on average, female students have done better in the classroom test. Such category statistics can be very helpful in getting some key insights, if you have categorical data in the data set. Now let's take a demo in which we will try to find a few category wise statistics in the Iris dataset.

Demo: Category Statistics

Here, we are in R studio. On line number 1, we are first loading the datasets package. Then we are importing the Iris dataset using the data command on line number 2. Next, we want to use a very useful function to get category wise statistics. The function describeBy, is a very handy function to get the category wise statistics in a tabular form. However, this function is available in the psych package. So we will first install the psych package using install. packages command as seen here on line number 3 then we will load this package using library command, as shown here on the line number 4. After loading the site package we can use the describeBy function. Inside this function the first parameter is the data set, that can be a matrix or a data frame. So we have passed the Iris data set, as the value of the first parameter. Then we can specify the group

parameter, this parameter represents which field you want to use to group the data. And as we want species-wise grouping, that is why have we set the species column to be the value of group parameter. So now let's execute lines from 1 to 5. And here on the console we have our result. So as you can see, we have values of various statistical indicators per category, or per group. So here we have the means sepal length for setosa species. Here we have the mean sepal length for versicolor species and here we have the mean sepal length for virginica species. Similarly for other columns also the mean value per category is available, other than mean this function returns other important statistical indicators also. Such as max, min, median, standard deviation, and some more advanced statistical values. So in this manner you can use the describe by function to get the per category statistics. Now let's look at few per category plots. We will start with histograms. Suppose you want to see the per category of the histogram of sepal length column. So we want to compare three histograms, where each histogram will belong to the sepal length distribution of one species or one category. Here we are going to use the histogram method available in the lattice package. We are first loading the lattice package and then we are using the histogram function. The first parameter may look a bit different to you. This is a formula format, or the formula interface. Remember that this formula is not like the Mathematical formula. Formula Format is a very neat technique in R which can help to specify columns as well as the function behavior behavior in a concise manner. Many R functions make use of the formula interface. However, interpretation of the formula may differ in different R functions. Therefore, make sure you understand the interpretation of the formula format for the particular R fuction. If you're not sure about the formula format for that R function, then you can go to the format documentation using the help command. Therefore this 2 gram function on line number 7 we have used the formula interface to specify the column to be used to create histogram. We can place the column name after the ___or release sign. So here we want to create the histogram on Sepal. Length column data on Iris Data Set. After the column name we have used the single pipe character. Typically this is used to represent the conditioning. Respecify the conditioning value will on the right hand side of the single pipe symbol. In this histogram function we want to condition the Sepal. Length column data on species factor variable. Roughly you can think this formula as Sepal. Length Column data, segregated by species column. Next for the data parameter we have passed the Iris Dataset We have also set a couple of extra parameters to format the resulting plot As we want to stack up all three histograms vertically. Therefore, we have set the layout parameter choosing an _____ _____. The first value of the _____ _____ signifies the number of clues, and the second value signifies the number of columns in the layout. We have also set the color of the histogram to be black. So know first load the lattice library and then

execute the histogram statement. So here we have our 3 vertically stacked histograms. So by looking into this plot we can see that on an average

Summary

So we have reached to the end of this module. In this module we learned various concepts related to exploring a given data set. So in most of the data analysis projects you will analyze continuous, or categorical data in a given data set. So first we learned the difference between these two types, then we delved into various techniques to explore and analyze the data set. We first started with the overall structure. And learned the use of str function, to get the number of observations, as well as number of features and their data types. We also learned the use of head function to see the sample data set, after looking at the overall structure, we learned about various concepts and techniques related to analyzing continuous data. We started with central tendency to measure the central point of data. And we discussed two main indicators, of central tendency, first was mean and second was median. Then you discuss that central tendency alone cannot tell you the real picture of data, therefore we have to look into the spread of data also. And we discussed a number of statistical indicators and plots, to explore the spread of data. We discussed range, quartiles, box plots, histograms, variance, and standard deviation. After exploring continuous data, we moved towards categorical data, for categorical data, we first learn various R functions to get frequency distribution of various categories in the data. We used table function to get number of samples or rows per category. We also used the table. prop function to get relative proportion for each category. After frequency distribution we looked at few techniques to get per category statistics. We used the describe by function available in the side package, To get various key statistical indicative values by category. We also created histograms and box plots, to see the spread of data per category. So this was all about exploring data in R. I'm sure concepts discussed in this module, will help you to explore and analyse your data in a better fashion.

Course author

Abhishek Kumar

Abhishek Kumar is a data science consultant, author, and Google Developers Expert (GDE) in machine learning. He holds a master's degree from the University of California, Berkeley, and has been...

Course info

Level	Beginner
Rating	★★★★½ (612)
My rating	★★★★★
Duration	7h 0m
Released	18 Oct 2014

Share course