Introduction to R

Introduction

Welcome to R! This document will walk you through everything you need to know to get started using R.

As you go through this module (and all future modules), please read slowly and carefully, and pay attention to detail. Every step depends on the correct execution of all previous steps, so reading quickly and casually might come back to bite you later in the assignment.

What is R?

R is a programming language specifically designed for doing statistics. Don't be intimidated by the word "programming" though. The goal of this course is not to make you a computer programmer. To use R to do statistics, you don't need know anything about programming at all. Every module throughout the whole course will give you examples of the commands you need to use. All you have to do is use those example commands as templates and make the necessary changes to adapt them to the data you're trying to analyze.

The greatest thing about R is that it is free and open source. This means that you can download it and use it for free, and also that you can inspect and modify the source code for all R functions. This kind of transparency does not exist in commercial software. The net result is a robust, secure, widely-used language with literally tens of thousands of contributions from R users all over the world.

R has also become a standard tool for statistical analysis, from academia to industry to government. Although some commercial packages are still widely used, many practitioners are switching to R due to its cost (free!) and relative ease of use. After this course, you will be able to list some R experience on your résumé and your future employer will value this. It might even help get you a job!

RStudio

RStudio is an "Integrated Development Environment," or IDE for short. An IDE is a tool for working with a programming language that is fancier than just a simple text editor. Most IDEs give you shortcuts, menus, debugging facilities, syntax highlighting, and other things to make your life as easy as possible.

We will be using a cloud version of RStudio called RStudio Server that we host here on campus.[1] You can access it using any web browser at

https://rstudio.westminstercollege.edu

Go to that web address and log in using your Westminster username and password (the same ones you use for your email, WebAdvisor, Canvas, and anything else you access). You'll want to bookmark that link for convenience.

Take a look around RStudio Server

Now that you're logged in to RStudio Server, let's explore some of the areas you'll be using in the future.

On the left side of your screen, you should see a big pane called the "Console". There will be some startup text there, and below that, you should see a "command prompt": the symbol ">" followed by a blinking

[1] Although it is possible to install R and RStudio on your own device, please use RStudio Server in the cloud for this course. Of course, if you want to continue to use R after you graduate, eventually you'll need to install it on your personal device. We can help you do this once this course is done, but we cannot provide technical support for running R on your own machine.

cursor. (If the cursor is not blinking, that means that the focus is in another pane. Click anywhere in the Console and the cursor should start blinking again.)

A command prompt can be one of the more intimidating things about starting to use R. It's just sitting there waiting for you to do something. Unlike other programs where you run commands from menus, R requires you to know what you need to type to make it work.

We'll return to the Console in a moment.

Next, look at the upper-right corner of the screen. There are two tabs in this pane. One is called "Environment" and the other is called "History". The "Environment" (also called the "Global Environment") keeps track of things you define while working with R. There's nothing to see there yet because we haven't defined anything! The "History" tab will likewise be empty; again, we haven't done anything yet.

Now look at the lower-right corner of the screen. There are five tabs here: "Files", "Plots", "Packages", "Help", and "Viewer". The "Files" tab will eventually contain the files you upload or create. "Plots" will show you the result of commands that produce graphs and charts. "Packages" will be explained later. "Help" is precisely what it sounds like; this will be a very useful place for you to get to know. We will never use the "Viewer" tab, so don't worry about it.

Try something!

So let's do something in R! Go back to the Console and at the command prompt (the ">" symbol with the blinking cursor), type

```
1+1
```

and hit Enter.

Congratulations! You just ran your first command in R. It's all downhill from here. R really is nothing more than a glorified calculator.

Okay, let's do something slightly more sophisticated. It's important to note that R is case-sensitive, which means that lowercase letters and uppercase letters are treated differently. Type the following, making sure you use a lowercase `c`, and hit Enter:

```
x <- c(1, 3, 4, 7, 9)
```

You have just created a "vector". When we use the letter `c` and enclose a list of things in parentheses, we tell R to "combine" those elements. So, a vector is just a collection of data. The little arrow `<-` says to take what's on the right and assign it to the symbol on the left. The vector `x` is now saved in memory. As long as you don't terminate your current R session, this vector is available to you.

Check out the "Environment" pane now. You should see the vector `x` that you just created, along with some information about it. Next to `x`, it says `num`, which means your vector has numerical data. Then it says `[1:5]` which indicates that there are five elements in the vector `x`.

At the command prompt in the Console, type

```
x
```

and hit Enter. Yup, `x` is there. R knows what it is. You may be wondering about the `[1]` that appears at the beginning of the line. To see what that means, try typing this (and hit Enter—at some point here I'm going to stop reminding you to hit Enter after everything you type):

```
y <- letters
```

R is clever, so the alphabet is built in under the name `letters`.

Type

```
y
```

Now can you see what the `[1]` meant above? Assuming the letters spilled onto more than one line of the Console, you should see a number in brackets at the beginning of each line telling you the numerical position of the first entry in each new line.

Since we've done a few things, check out the "Global Environment" in the upper-right corner. You should see the two objects we've defined thus far, `x` and `y`. Now click on the "History" tab. Here you have all the commands you have run so far. This can be handy if you need to go back and re-run an earlier command, or if you want to modify an earlier command and it's easier to edit it slightly than type it all over again. To get an older command back into the Console, either double-click on it, or select it and click the "To Console" button at the top of the pane.

When we want to re-use an old command, it has usually not been that long since we last used it. In this case, there is an even more handy trick. Click in the Console so that the cursor is blinking at the blank command prompt. Now hit the up arrow on your keyboard. Do it again. Now hit the down arrow once or twice. This is a great way to access the most recently used commands from your command history.

Let's do something with `x`. Type

```
sum(x)
```

I bet you figured out what just happened.

Now try

```
mean(x)
```

What if we wanted to save the mean of those five numbers for use later? We can assign the result to another variable! Type the following and observe the effect in the Environment.

```
m <- mean(x)
```

It makes no difference what letter or combination of letters we use to name our variables. For example,

```
mean_x <- mean(x)
```

just saves the mean to a differently named variable. In general, variable names can be any combination of characters that are letters, numbers, underscore symbols (`_`), and dots (`.`). (In this course, we will prefer underscores over dots.) You cannot use spaces or any other special character in the names of variables.[2] You should avoid variable names that are the same words as predefined R functions; for example, we should not type `mean <- mean(x)`.

Load Packages

Packages are sets of commands and functions that people all over the world write. These packages extend the capabilities of R and add useful tools. For example, we would like to use the `MASS` package because it includes an interesting data set on patients with malignant melanoma.

The data set is called `Melanoma`. (Again, remember that R is case-sensitive, so make sure you use a capital `M` in `Melanoma`.) Let's see what happens when we try to access this data set without loading the package that contains it. Try typing this:

```
Melanoma
```

You should have received an error. That makes sense because R doesn't know anything about a data set called `Melanoma`.

[2] The official spec says that a valid variable name "consists of letters, numbers and the dot or underline characters and starts with a letter or the dot not followed by a number."

Now type this at the command prompt:

```
library(MASS)
```

It didn't look like anything happened. However, in the background, all the functionality of the `MASS` package became available to use.

Let's test that claim. Hit the up arrow twice and get back to where you see this at the Console (or you can manually re-type it, but that's no fun!):

```
Melanoma
```

Now R knows about the `Melanoma` data, so the last command printed it all to the Console.

Go look at the "Packages" tab in the pane in the lower-right corner of the screen. Scroll down a little until you get to the "M"s. You should be able to find the `MASS` package. You'll also notice a check mark by it, indicating that this package is loaded into your current R session.

You must use the `library` command in every new R session in which you want to use a package. If you terminate your R session, R forgets about the package. If you are ever in a situation where you are trying to use a command and you know you're typing it correctly, but you're still getting an error, check to see if the package containing that command has been loaded with `library`. (Many R commands are "base R" commands, meaning they come with R and no special package is required to access them. The set of `letters` you used above is one such example.)[3]

Getting help

There are four important ways to get help with R. The first is the obvious "Help" tab in the lower-right pane on your screen. Click on that tab now. In the search bar at the right, type `Melanoma` and hit Enter. Take a few minutes to read the help file.

Help files are only as good as their authors. Fortunately, most package developers are conscientious enough to write decent help files. But don't be surprised if the help file doesn't quite tell you what you want to know. And for highly technical R functions, sometimes the help files are downright inscrutable. Try looking at the help file for the `grep` function. Can you honestly say you have any idea what this command does or how you might use it? Over time, as you become more knowledgeable about how R works, these help files get less mysterious.

The second way of getting help is from the Console. Go to the Console and type

```
?letters
```

The question mark tells R you need help with the R command `letters`. This will bring up the help file in the same Help pane you were looking at before.

Sometimes, you don't know exactly what the name of the command is. For example, suppose we misremembered the name and thought it was `letter` instead of `letters`. Try typing this:

```
?letter
```

You should have received an error because there is no command called `letter`. Try this instead:

```
??letter
```

and scroll down a bit in the Help pane. Two question marks tell R not to be too picky about the spelling. This will bring up a whole bunch of possibilities in the Help pane, representing R's best guess as to what you might be searching for. (In this case, it's not easy to find. You'd have to know that the help file for `letters` appeared on a help page called `base::Constants`.)

[3]When using RStudio Server in the cloud, the packages that you need for this course (and many others) are pre-installed for you. If you ever need a package that isn't installed already, send an email to Sean Raleigh: sraleigh@westminstercollege.edu

The fourth way to get help—and often the most useful way—is to use your best friend Google. You don't want to just search for "R". (That's the downside of using a single letter of the alphabet for the name of a programming language.) However, if you type "R _____" where you fill in the blank with the topic of interest, Google usually does a pretty good job sending you to relevant pages. Within the first few hits, in fact, you'll often see an online copy of the same help file you see in R. Frequently, the next few hits lead to StackOverflow where very knowledgeable people post very helpful responses to common questions.

Use Google to find out how to take the square root of a number in R. Test out your newly-discovered function on a few numbers to make sure it works.

Understanding the data

Let's go back to the melanoma data contained in the `Melanoma` data set from the `MASS` package.

The first thing we do to understand a data set is to read the help file on it. (We've already done this for the melanoma data.) Of course, this only works for data files that come with R or with a package that can be loaded into R. If you are using R to analyze your own data, presumably you don't need a help file. And if you're analyzing data from another source, you'll have to go to that source to find out about the data.

Next, we can also look at the data in "spreadsheet" form. Type

```
View(Melanoma)
```

(Be sure you're using an upper-case "V" in `View`.) A new pane should open up in the upper-left corner of the screen. In that pane, the melanoma data appears in a grid format, like a spreadsheet. The observations (the patients in the study) are the rows and the variables are the columns.

Earlier when we typed `Melanoma`, it was a little annoying that R printed out the entire data set to the Console. For this data set, it was only a minor aggravation, but it could become a major problem if we're analyzing data sets with thousands (or even millions) of rows. Try this instead:

```
head(Melanoma)
```

We can customize this by specifying the number of rows to print. (Don't forget about the up arrow trick!)

```
head(Melanoma, n = 10)
```

The `tail` command does something similar.

```
tail(Melanoma)
```

We want to understand the "structure" of our data. For this, we use the `str` command. Try it:

```
str(Melanoma)
```

This tells us several important things. First it says that there are 205 observations of 7 variables. We can isolate those pieces of information separately as well, if we need:

```
NROW(Melanoma)
```

```
NCOL(Melanoma)
```

These give you the number of rows and columns, respectively.

The `str` command also tells us about each of the variables in our data set. We'll talk about these later.

We need to be able to summarize variables in the data set. The `summary` command is one way to do it:

```
summary(Melanoma)
```

You may not recognize terms like "Median" or "1st Qu." or "3rd Qu." yet. Nevertheless, you can see why this summary could come in handy.

There are only some of the variables for which this summary makes sense. For example, something is weird about taking the mean of the `status` variable. Think about what's wrong here. (Hint: look at the "spreadsheet" view and peruse the values of `status`. Remember the help file too!)

Understanding the variables

When we want to look at only one variable at a time, we use the dollar sign to grab it. Try this:

```
Melanoma$age
```

This will list the entire `age` column, in other words, the list of ages of the patients in this particular study. If we only want to see the first few, we can use `head` like before.

```
head(Melanoma$age)
```

If we want the structure of the variable `age`, we do this:

```
str(Melanoma$age)
```

Notice the letters `int` at the beginning of the line. That stands for "integer" which is another word for whole number. In other words, the patients' ages all appear in this data set as whole numbers. There are other data types you'll see in the future:

- `num`: This is for general numerical data (which can be integers as well as having decimal parts).
- `chr`: This means "character", used for character strings, which can be any sequence of letters or numbers. For example, if the names of the patients were recorded in the melanoma data, these names would be recorded in a character variable.
- `factor`: This is for categorical data. These are generally recorded like character strings, but factor variables have more structure because they take on a limited number of possible values corresponding to a generally small number of categories. We'll learn a lot more about factor variables in future modules.

There are other data types, but the ones above are by far the most common that you'll encounter on a regular basis.

If we want to summarize only the variable `age`, we can do this:

```
summary(Melanoma$age)
```

Working in the cloud

There is one aspect of working in the cloud that is a little different from your normal workflow. Because you are accessing RStudio Server through a web browser, nothing that you do is actually happening on your hard drive on your personal computer. Everything you see on screen, every command that you run, and every file that you create, access, edit, or save exists on a server somewhere else.

This makes it a little tricky to work with files that you have on your computer. In the next few sections, we'll practice by uploading a file that we'll need for the next module, which will cover the topic of using R Markdown.

Projects

Using files in R requires you to be organized. R uses what's called a "working directory" to find the files it needs. Therefore, you can't just put files any old place and expect R to be able to find them.

One way of ensuring that files are all located where R can find them is to organize your work into projects. Look in the far upper-right corner of the RStudio Server screen. Underneath your username, you should see

some text that says `Project: (None)`. This means we are not currently in a project. We're going to create a new project in preparation for the next module on using R Markdown.

Open the drop-down menu here and select `New Project`. You may be prompted to save your workspace. If you click "Save", that just means that the commands you've typed in this session will be available the next time you open up RStudio Server. It's really not a big deal now, but you might care in the future. So click either "Save" or "Don't Save".

When the dialog box opens, select `New Directory`, then `Empty Project`.

You'll need to give your project a name. In general, this should be a descriptive name—one that could still remind you in several years what the project was about. The only thing to remember is that project names and file names should not have any spaces in them. In fact, you should avoid other kinds of special characters as well, like commas, number signs, etc. Stick to letters and numerals and you should be just fine. If you want a multiword project name or file name, I recommend using underscores like this: `this_filename_has_spaces_in_it`.[4]

In this case, let's type `Using_R_Markdown` for the "Directory name". Leave everything else alone and click `Create Project`.

You will see the screen refresh and R will restart.

You will see a new file called `Using_R_Markdown.Rproj` in the Files pane, but **you should never touch that file**. It's just for RStudio to keep track of your project details.

If everything works the way it should, creating a new project will create a new folder, put you in that folder, and automatically make it your working directory.

You can see that a new folder was created. Near the top of the Files pane, you should see a "Home" icon and `Home > Using_R_Markdown` next to it. Click on the word "Home" here. This will take you to your Home directory. You probably didn't realize it, but while we were doing all that stuff in the first part of this module, you were sitting inside that Home directory. However, at no point did we save any files, so there is nothing sitting inside your Home directory except maybe some technical internal files (these tend to start with a period) and a new folder called `Using_R_Markdown` that you just created. Click on that `Using_R_Markdown` folder to get back into it. You should see the file `Using_R_Markdown.Rproj` sitting there still. (Again, **do not touch that file**. Do not click on it. Do not try to open it. Do not try to edit it. It's an internal file used to keep track of your project, that's all.)

Uploading files

Before you can upload a file to RStudio Server, you need to have the file available on your personal computer. In a separate tab of your web browser, navigate to the website where your module files are stored by your professor. (For most of you, I imagine this will be Canvas.) With your professor's help, locate the file called `Using_R_Markdown.Rmd`. When you click on it, most browsers will download the file to your computer, typically to a folder called "Downloads" somewhere on your machine. The location of this folder varies between Windows and Macs, and even varies depending on which version of the aforementioned operating systems you happen to be using. For the next step, you'll need to know where that file ended up, so using "File Explorer" or the "Finder" or whatever application you use on your machine to browse files, make sure you know where to find the file you just downloaded. Request help from someone knowledgeable about your operating system if you need.

Now go back to the window or tab containing RStudio Server. Make sure you're still in the new project. (The upper-right hand corner of the screen should say `Using_R_Markdown` under your username.) And make sure you're in the project folder. (In the Files pane, make sure that next to the Home icon it says `Home > Using_R_Markdown`. You should also be able to see the file `Using_R_Markdown.Rproj`.)

[4] R will let you name projects with spaces and modern operating systems are set up to handle file names with spaces, but there are certain things that either don't work at all or require awkward workarounds when file names have spaces.

Look for the "Upload" button in the toolbar of the Files pane. It will be right above where it says `Home > Using_R_Markdown`. Click the "Upload" button. A dialog box will appear, and you need to click "Choose File". This will open yet another window that will allow you to navigate to your Downloads folder and find the file you downloaded called `Using_R_Markdown.Rmd`. Again, ask for help if you're having trouble finding the file. Once you've selected the file, click "Open". You will go back to the initial dialog box where you'll be able to click "OK". The file will appear in your project directory.

We won't do anything with this file for now. That will be the topic of the next module, called (as you might imagine) "Using R Markdown".

Conclusion

It is often said that there is a steep learning curve when learning R. This is true to some extent. R is harder to use at first than other types of software. Nevertheless, in this course, we will work hard to ease you over that first hurdle and get you moving relatively quickly. Don't get frustrated and don't give up! Learning R is worth the effort you put in. Eventually, you'll grow to appreciate the power and flexibility of R for accomplishing a huge variety of statistical tasks.

Onward and upward!