Exploratory Data Analysis

Lesson 2: R Basics - Notes

Quick links

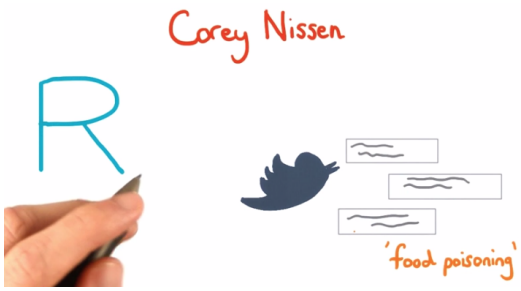
The Power of R

Intro to Lesson 2

Welcome to lesson two. In this lesson, we'll learn about the advantages of using R to perform data analyses. We'll install R Studio. And we'll learn the basic data types and commands of R.

Analyzing Tweets in Chicago

Before we get started, I want to share with you how one data scientist used R to improve the health outcomes for the citizens of Chicago, Illinois. Data scientist Corey Nissen used the R programming language to analyze tweets from people in the city of Chicago. Corey analyzed tweets in real time by searching for the words food poisoning in each of them. I want you to read more about his analysis and his work by clicking on the link in the instructor notes. Try to find the answers to these next three questions as you read through the article.



Quiz

What was the name of the R package used by Corey? How many Tweets per day did the system flag? (lower and upper limits) What did you find most interesting about the article?

Answers

Textcat was the name of the R package that Corey used. We'll discuss R packages in the next video and what they allow us to do. For the second question, the automated system captures between ten tweets to 20 tweets a day. For each of the tweets, the system will recommend that whoever sent the tweet file a report. Now, for this last question, we accepted any text answer here.

I thought that the most interesting part was that Cory showed the open source R code classifier on GitHub. GitHub is a repository for sharing code. You can learn more about GitHub by following the links in the instructor notes. It's the most popular repository for open source projects and it's really easy to get started.

Why R?

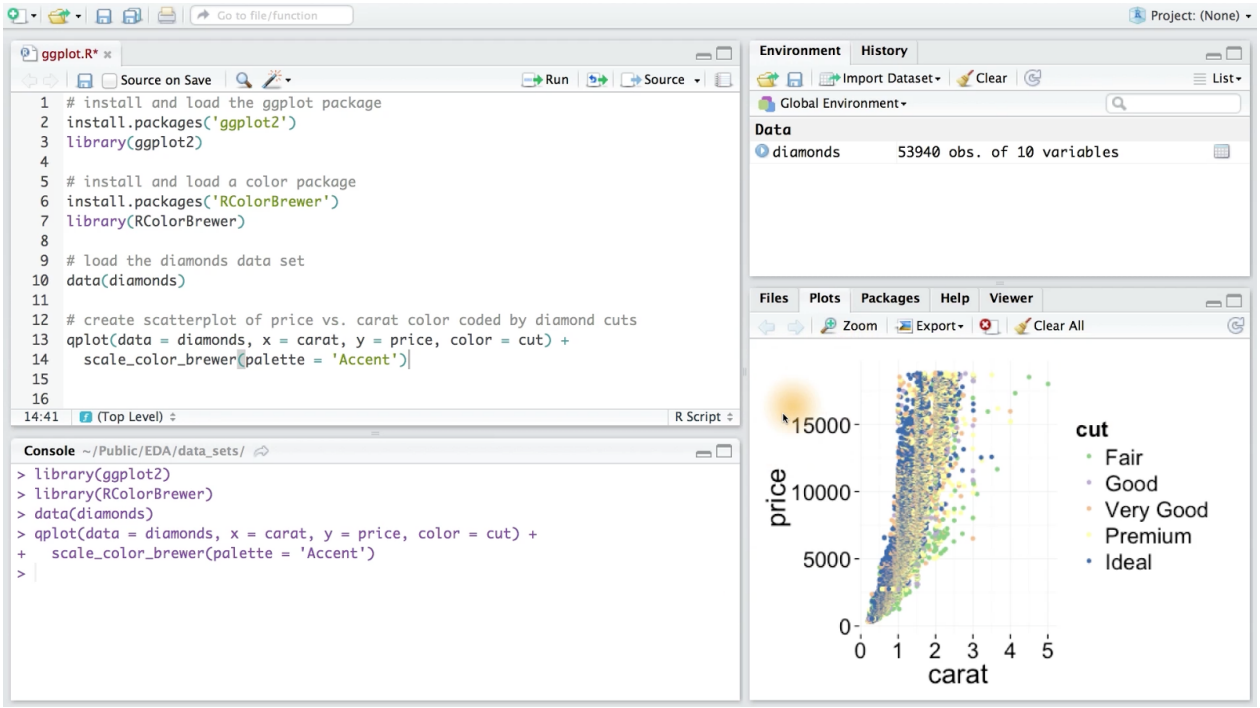
About R

R is the leading programming language for statistics and data analysis. One of the main advantages of using R is that we can build up an analysis line by line in code. We can save all of our work in a file and go back to see what we investigated at a later date. Having R scripts allows you to easily share your work with others. And you can see what others are doing with data. R also has over 2,000 user contributed packages that increase its functionality. One example of this is the text analysis package, TextCat, that you just saw.

ggplot2

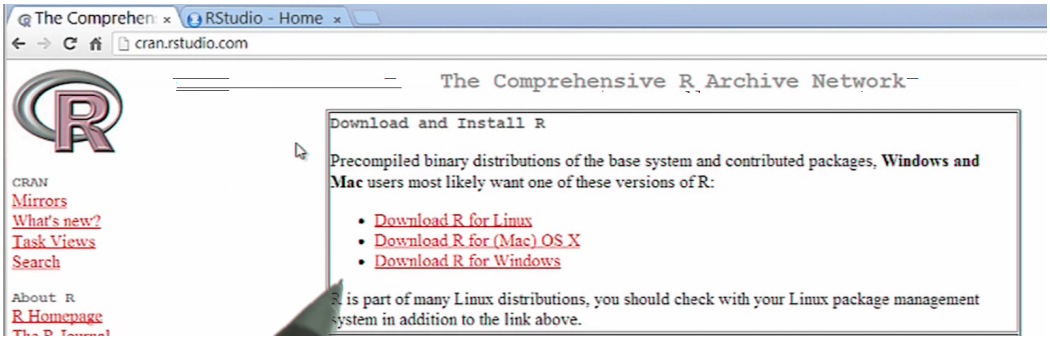
Now, one of the main packages we'll use throughout this course is ggplot2. ggplot2 is a graphics package that lets us create plots and graphs with just a few commands. We'll learn more about ggplot in the next lesson but just to give you a taste of it, here's an example of a plot you can create. Don't worry about memorizing or understanding all of this code. You'll have plenty of practice with this later in the course. I just want to show you how a few lines of code, can create amazing graphics.

I'm going to load up the ggplot library and a color library. Then I'm going to load the diamonds data set, and with this function I'll create a scatter plot. Let's check out



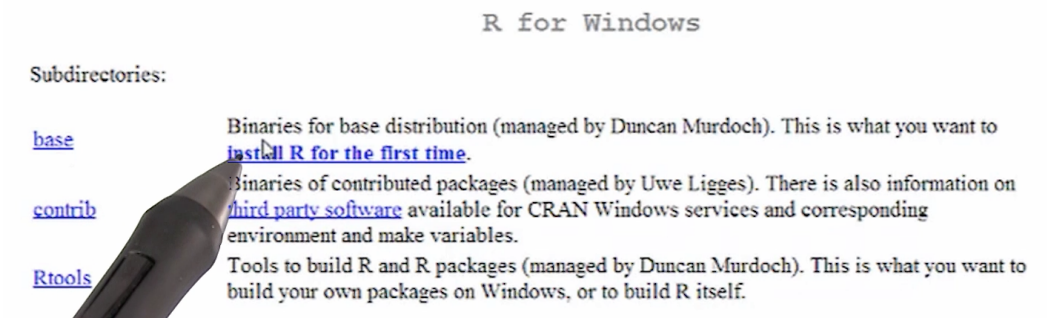
this plot in detail. This part shows the relationship between price and carat of almost 54,000 round cut diamonds. I'd say R is doing very well for such few lines of code. The last thing I want to mention is that you can use R anywhere. It's free open source software that works on any operating system. And as a result of this, R has a large active and growing community of users.

Install RStudio on Windows

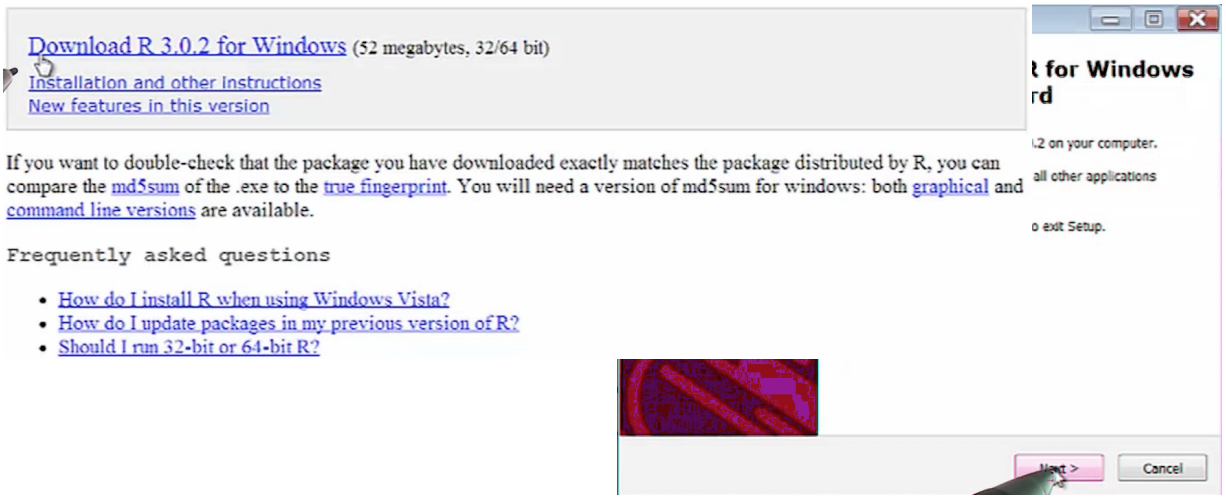


R Programming Language Installation

If you're using windows, I'll show you how to install R and R Studio. To get started you want to go to cran.rstudio.com. And then you'll click on the link to download R for Windows. If you never installed the R programming language before, then click on this first link. Since it's your first time downloading it. Now depending on your version of Windows, you might need to see the frequently asked questions. I'm set for my computer, so I'm going to go ahead and download R for Windows. I get a warning message and I want to keep the file. Now this might take some time to download, so just be patient. Then you want to open the file, and then run the program. A set up wizard should appear and now you just want to walk through all the steps. I'll set my language, and then I'll just continue installing by hitting next. I'm just going to accept the defaults here and then install it to R. I'll leave these as is, so that way a desktop

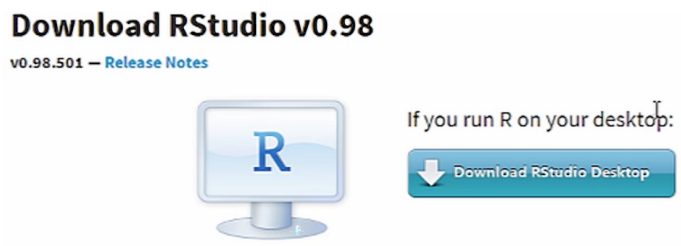


icon will be created. And it's going along. Great. Now I'm finished with my setup.

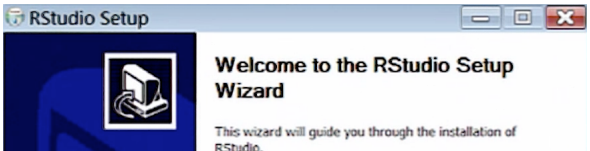


RStudio Installation

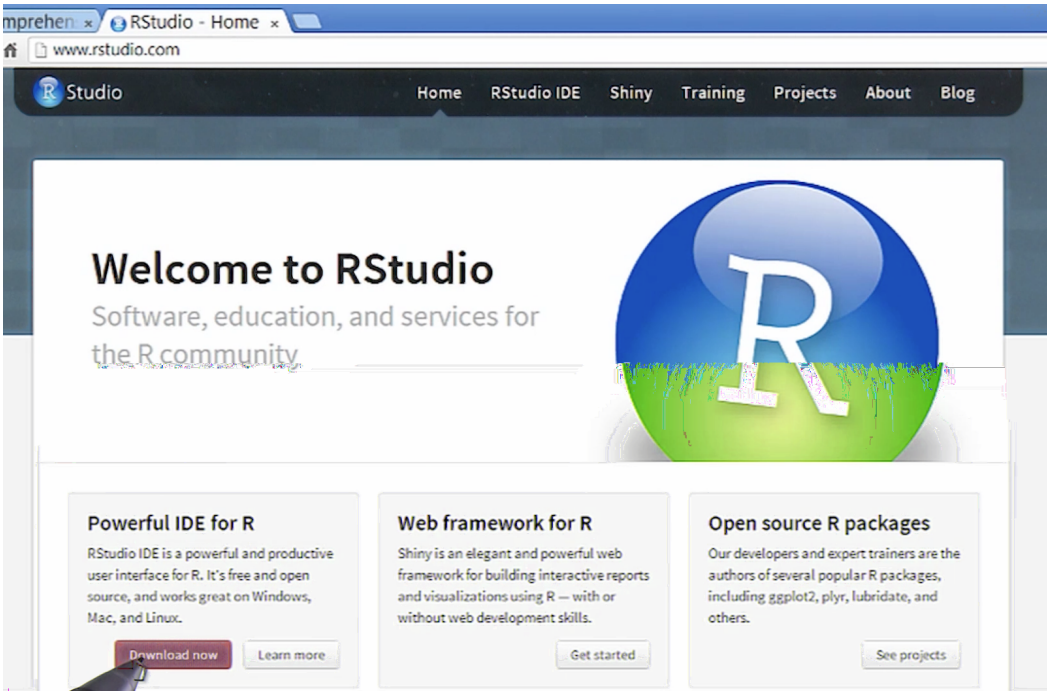
So that was just part one of the installation process. That just gave us the



programming language R. Now we need to go and install RStudio. RStudio is a graphical user interface for programming in R. It's basically a point and click system that lets us work with data really easily. So first let's just go to download now to get RStudio. We're going to get RStudio for our desktop. So just click this first button. Now there's going to be a lot of links here, and you really just want this first link. It's going to be the version of RStudio that's recommended for your computer, and for

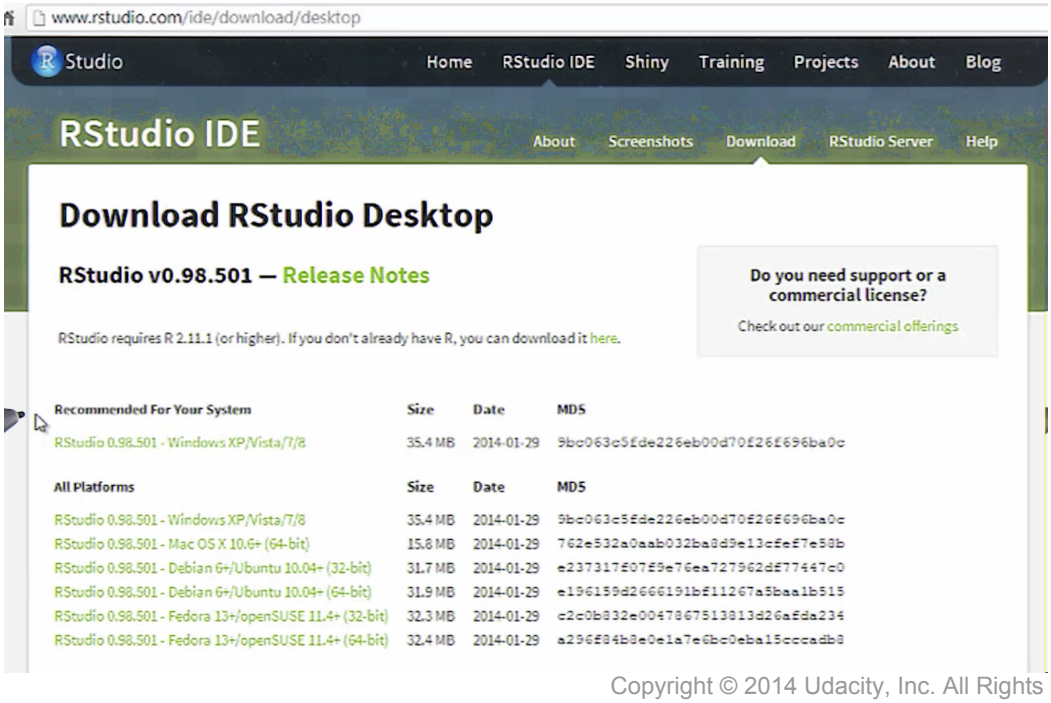


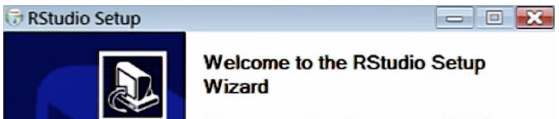
your operating system. Notice how this one's for Windows XP, Vista, Windows 7, or Windows. Now that it's ready, so I'm going to open this up, and I get the wizard you can see on the right. I'll follow the instructions, then install RStudio into this path. And create a Start Menu folder as well. Great. I'm done. And this is what RStudio looks like. Now yours might look slightly different than mine and that might be because the file's open. So if I go to File > New File > RScript this opens a basic RScript



which allows me to start programming in R.

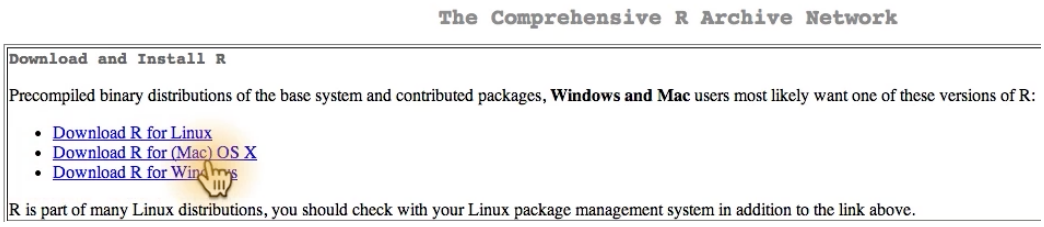
Install RStudio on a Mac



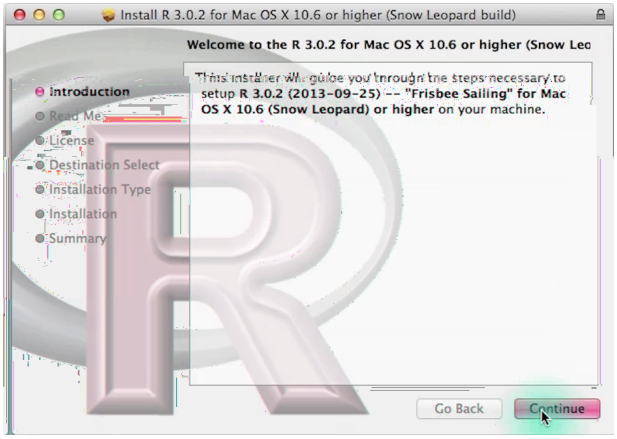


In this video, I'll show you how to install R Studio on a Mac, which is a graphical interface for programming with the R language. Simply open up any web browser and go to RStudio.com then you want to go to download now. You are going to run R Studio from your desktop so click on this link. Now here's where I want you to be very careful. R studio requires R first, so if you don't already have the programming language R installed, you need to download it here. Now this is a really easy link to miss so make sure you click here first to install R, before installing R studio. Once you're at cran.rstudio.com, you want to select the download for the Mac OSX version. This will bring up another page and then you can download the latest version of R. It may take some time to download so you might want to take a break and then come back to the computer.

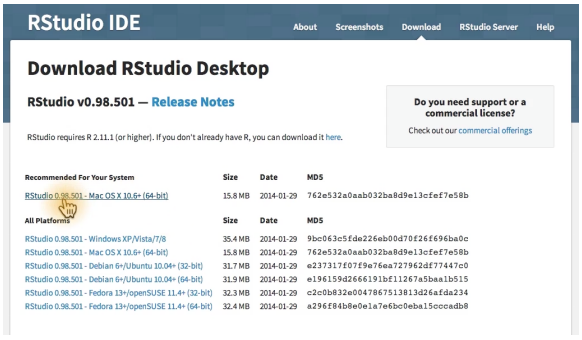
Once you've opened up the package, follow the instructions for installing R. You should end up getting a successful message for the installation. Now you're ready to go and install R studio. This website will recommend the best version of RStudio for you so just click on whatever one comes up for you first. Once you've downloaded this, you want to open that package. Now, just drag RStudio into your applications

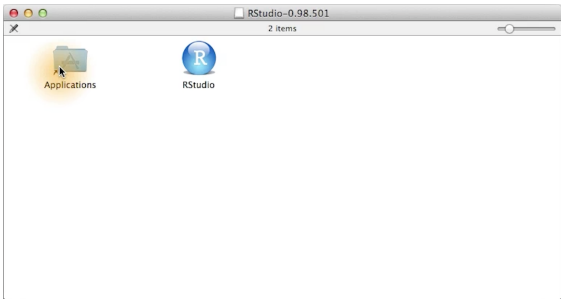


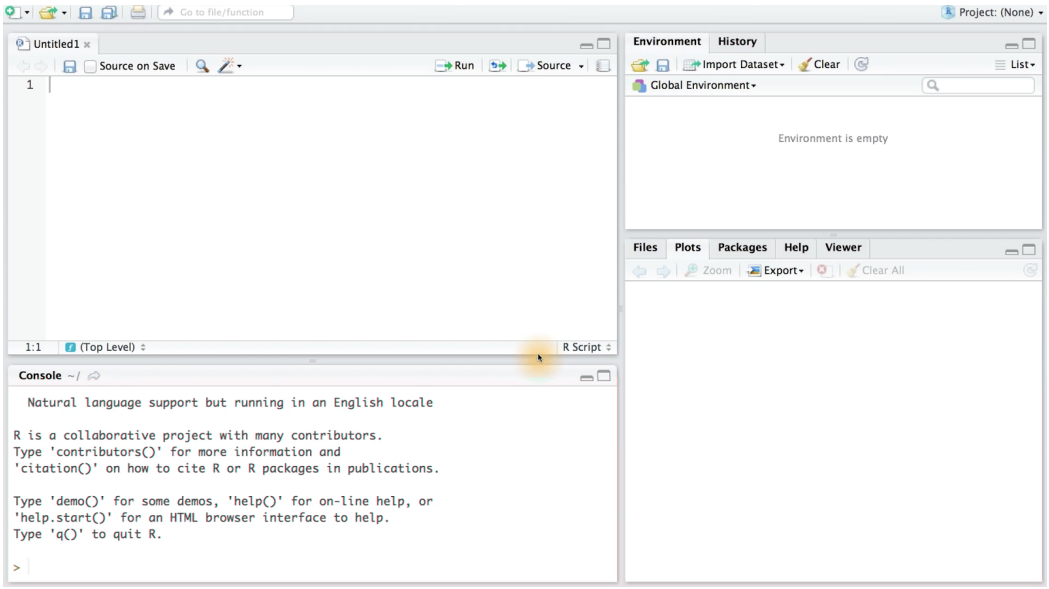
and you'll be good to go.



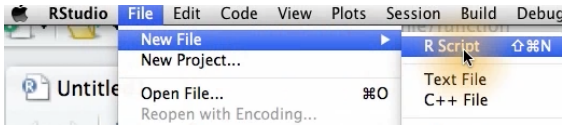
RStudio Layout

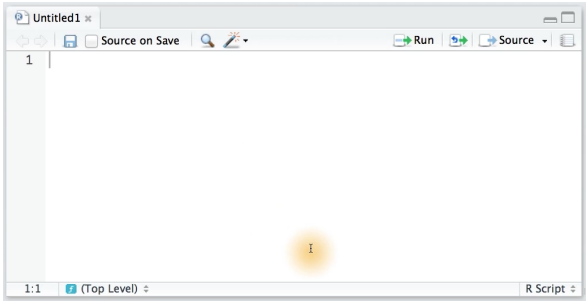






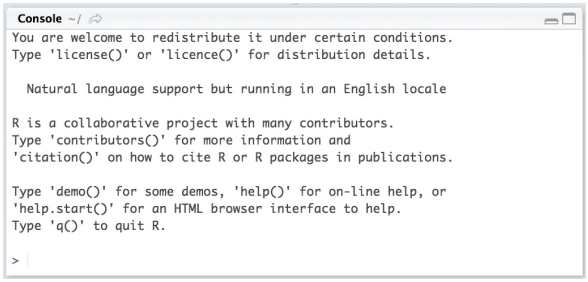
This is R studio, it's an IDE or an Integrated Development Environment and it's what we'll be using to create R scripts and to create plots. It's a



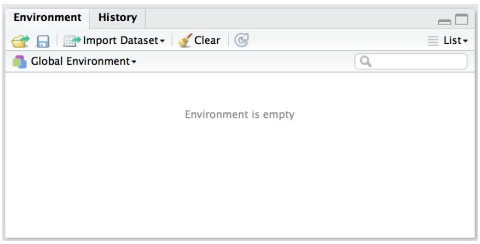


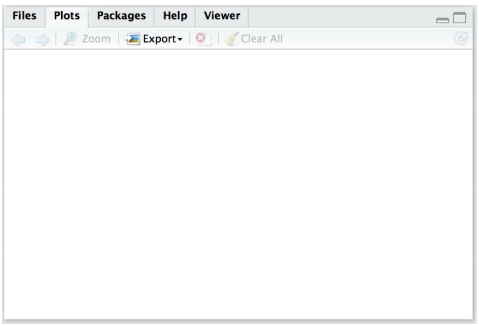
point and click system that organizes our workspace and it has some nice features for creating and sharing work. Here's a basic tour of it. The main interface contains four panels that can be resized as needed. Now, you may only see three panels when you open R studio. If so, just open a new R script by clicking on the file, and then New File > Rscript.

So, in the top left here, we have Rscripts. These are the files in which we will type our R commands. We'll also save these files, so we can alter them or share them at a later time. In the bottom left, we have the R Console. You can type in R commands here. However, these R commands will only be saved in a temporary history. The commands won't be saved to a file like up in here that can be reviewed at a later date.



In the top right of R studio, we have two tabs. One called Environment and one called History. The Environment will contain all the objects, functions and values that are in the current working memory for R. History on the other hand will keep a running log of any of the R commands that we run. Now we can run R scripts directly from our files but we can also type them into the console. No matter which way we type them in the History will capture them.

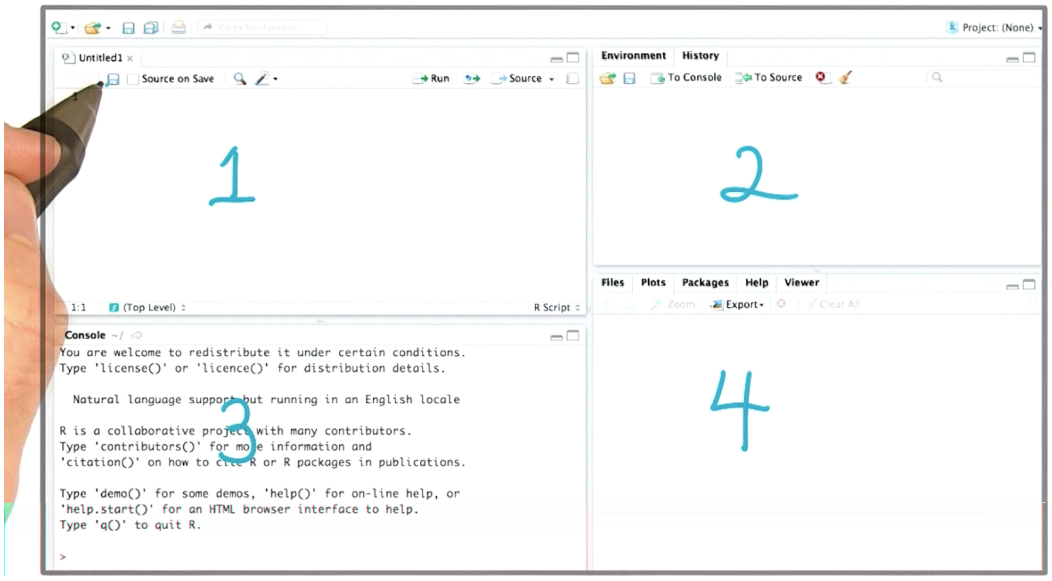




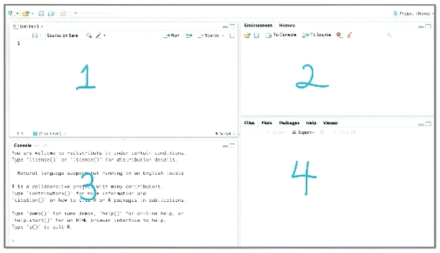
And finally, in the lower right, we have the Files, Plots, Packages, Help and Viewer section. You'll learn more about these later but this is where our plots or graphs will first appear when we run an R command that creates a visualization.

Quiz

Now that we've gone over the layout of R Studio, I want you to match some actions that you might do in each panel of R. Remember this first panel is for R files or Rscripts. This second area is for the workspace or the history. This third space is



the Console where we get our output. And this fourth panel is for Files, Packages, Plots, and Help. So for example, if I asked you to match the activity to one of these panels, like save an R script that would correspond to box number one. You can save them by clicking on this button. Now we could of course use the main menu, but I'm actually include as any one of your options here. The main purpose of this quiz is just to get you familiar with R studio. So here are your list of actions. And I want you to pair it with the panel where you would do it in R studio. Put the number of the corresponding panel in each of the boxes.



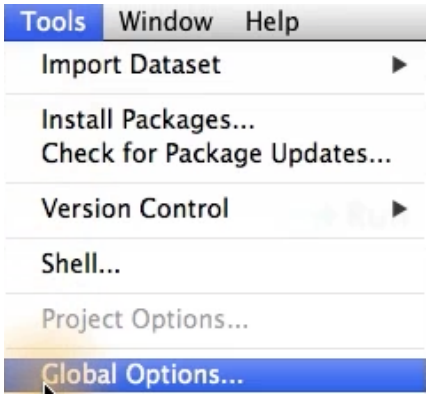
1 R scripts / files
2 Environment / History
3 Console
4 Files / Plots / Packages / Help

- [x] 1 Save an R script
- [ ] Review the log of commands entered
- [ ] Read help documentation
- [ ] Clear the workspace
- [ ] Run a bunch of commands from a file
- [ ] Look at a plot
- [ ] See a list of objects in memory
- [ ] Read the results from functions or calculations

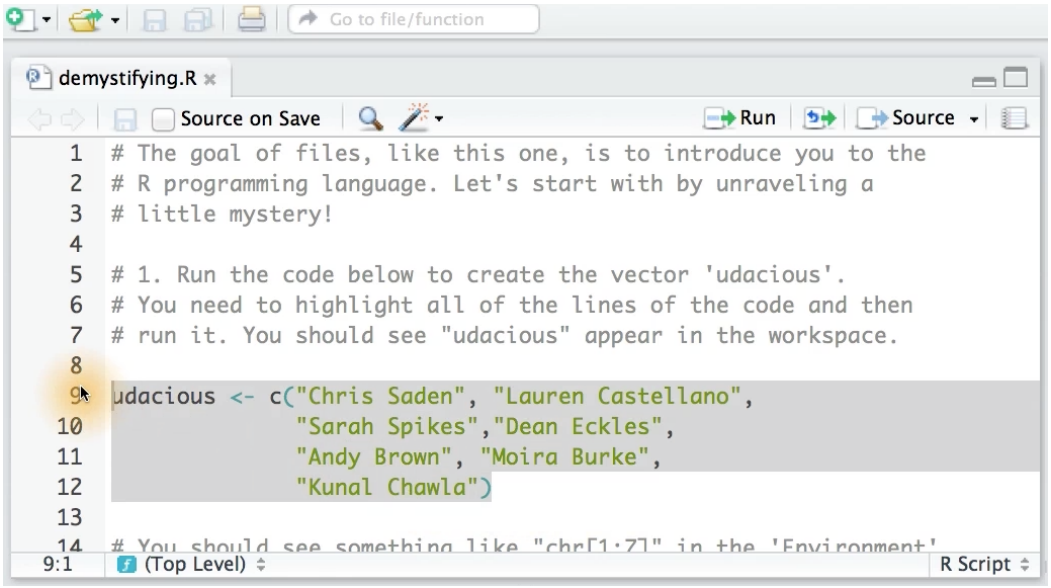
Answer

You can review the log of command entered by going to the history so that would be panel two. Help documentation appears in the lower right hand side which would be panel four. To clear the workspace you can either use panel two or panel three. You can run a command in here to clear the workspace or you can use the icon located in panel two. It should look like a sweeping broom. We can run a bunch of commands from a file from panel one. Plots appear down here in the lower right, so that would be panel four, and all of our objects and memory appear up here, in our environment or our work space. So that's panel two. And finally to read the results from functions or from calculations, we simply look at the console, where our output comes, panel three.

Demystifying R



Now that you know the basic layout of our studio, you're welcome to customize it, by going to Tools, and then Options. From here you can choose the default directory if you'd like. You can change formatting options for editing code, or you can even change the font and the appearance. Here, I'm using the tomorrow theme, but there are of course some others. You can also make adjustments to the panels so, feel free to play around with this to get a configuration that you like. Now, you might not have any idea what you like first so just wait until you play with r a bit, and then you'll find a configuration that suits you. I'm going to leave this as is.



Alright, we've got R set up, so let's start using the programming language. To get started, I want you to download the R script (demystifying.R) and open it here. When you open the file, it should look like this. This R script contains both comments as plain text and R code. You can run R commands by highlighting a chunk of the code and then hitting Cmd+Enter on a Mac, Or Ctrl+Enter on a PC. The lines of text that start with a hash or a pound symbol are comments. There lines are plain text that we don't want to run as R code. It's the standalone text that you really want for the R code.

Now, your task is to read through this file and run and write code when you're prompted to do so. When you get to the end of the file, you'll get to a question, and you'll need to answer this question to move on to the next video in this lesson. Now, if you're already familiar with R, I suggest trying to answer the questions, and then you might want to go back to the files to see if you missed anything.

We know you learn best by doing, so it's important that you execute and run this code as you go, and that you make sense of the output. Any output will appear down here in the console. I also hope that you have moments of inspiration. So if something does come to mind I want you to code it up and just run it in R. You won't break anything and the worse thing that would happen would be an error message or a warning message in this box. If you do happen to get stuck at any point post in the discussions so that way your peers or one of the instructors can help.

Quiz

Download the demystifying.R file and open it in R Studio. As you read through it, read and write code when prompted. When you get to the end of the file answer this question: what is the average mpg (miles per gallon) for all of the cars in the mtcars dataset?

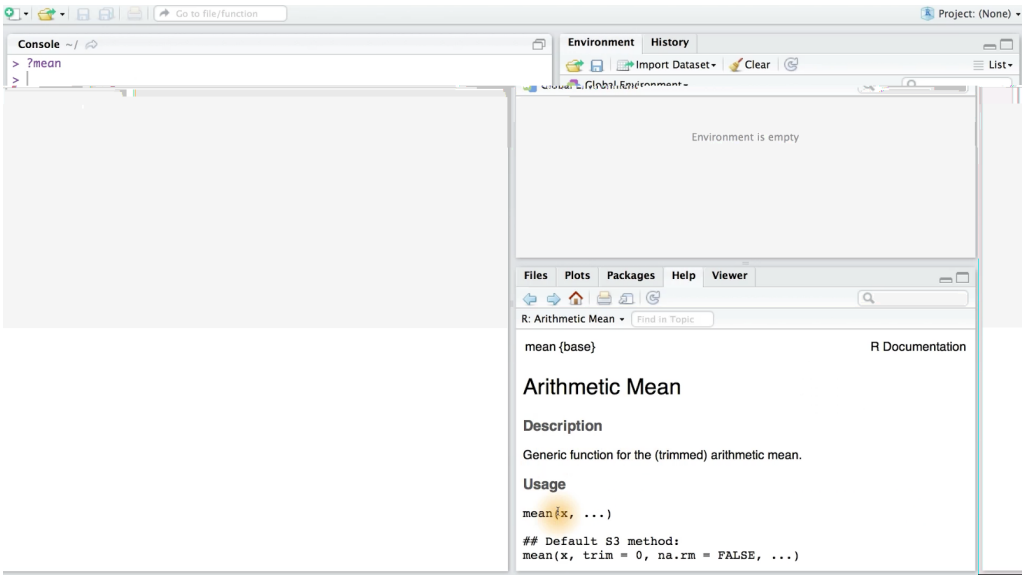
Answer

This question should have been pretty simple. You just needed to run this bit of code, "**mean(mtcars$mpg)**", and that would have given you the value to enter.

Getting Help

As you go through this course, you might encounter commands that you've never seen before. And we don't expect you to know everything, but we do want you to keep the following in mind when you encounter any sort of challenges or difficulties. First, take an active role in problem solving and be aware of the resources at your disposal. You can type a question mark and the name of any function to bring up help documentation in R.

So for example I can look up the documentation for the function mean. Here it will tell me any parameters that mean takes. And it will give me examples usually at the bottom of the documentation. I can copy and paste these into the console. And press enter to see what the function does. Here, I created something called **x**, and then if I hit **x** into the console, I can see what this is. This is just the vector of numbers zero to ten, and then the number 50. I can take the mean of **x**, and print it out. And finally, I have some sort of trend



parameter. Maybe you can figure out what this does.

```
Console ~/
> ?mean
> x <- c(0:10, 50)
> x
 [1]  0  1  2  3  4  5  6  7  8  9 10 50
> xm <- mean(x)
> xm
[1] 8.75
> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

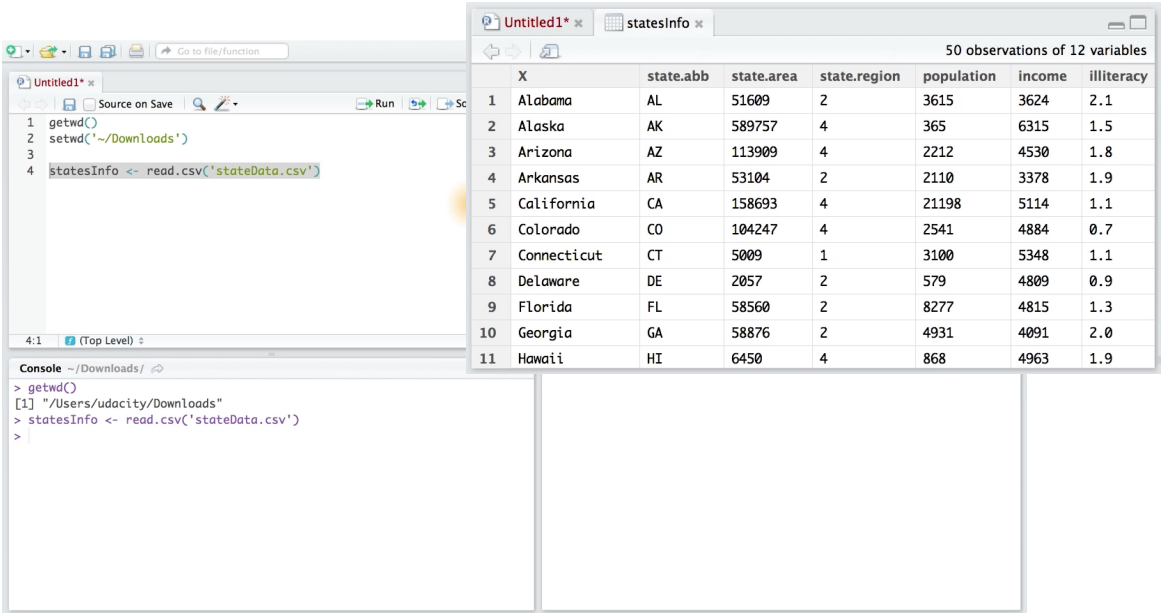
Now, if you can't solve your problem using the documentation I would suggest Googling it. StackOverflow is also another great site to browser for help and for an FAQ. One of my personal favorites when I learned R was Quick-R, or statmethods.net. This site has excellent examples and short pieces of code that can refresh your memory and get you going on any data analysis. I've linked to a couple other websites like the R Cookbook and R-bloggers, and you can also find these in the course materials under R Resources on the course wiki.

Read and Subset Data

In the next three lessons, we will explore a data set that resembles user activity on Facebook but before we get to that data set I need to show you how to read in data that doesn't come with R and how to subset that data. It turns out that not all data sets can be easily loaded like the mtcars data set. You can find a link to a new data set here. This data set is going to contain information on the fifty states in the US. Most of the data comes from about 1970.

Before we can read in the data, we need to set our current working directory. So, to figure out what directory you're in now, you can type in **getwd()**. We can run

this command and see the output to the console. It looks like I'm already in the downloads file, and that also appears here in the top of my console. So I don't need to change my directory, but maybe you do. To change your directory, you can type **setwd('***directory***')**. This will take a string which will be the file path to whatever directory you want to go to. My guess is that your data set is in your downloads file. So I would probably run this command (**setwd('~/Downloads')**). Now it's important to note that whether or not you are on a Mac or Windows machine you still need to separate your paths or your folders with a forward slash. Also be sure that you use quotes around your path.



	X	state.abb	state.area	state.region	population	income	illiteracy
1	Alabama	AL	51609	2	3615	3624	2.1
2	Alaska	AK	589757	4	365	6315	1.5
3	Arizona	AZ	113909	4	2212	4530	1.8
4	Arkansas	AR	53104	2	2110	3378	1.9
5	California	CA	158693	4	21198	5114	1.1
6	Colorado	CO	104247	4	2541	4884	0.7
7	Connecticut	CT	5009	1	3100	5348	1.1
8	Delaware	DE	2057	2	579	4809	0.9
9	Florida	FL	58560	2	8277	4815	1.3
10	Georgia	GA	58876	2	4931	4091	2.0
11	Hawaii	HI	6450	4	868	4963	1.9

Now in order to load up the data, we can use the **read.csv('***filename***')** command. This command takes a string, which is the name of the file. And here we're going to pass it to a variable called **statesInfo**. **statesInfo** is going to save all of our data into a data frame. When I run this code, I can see that states info appears in my environment. I can double click on the data frame in the workspace, and this will let me see the table of values in R Studio.

Now, let's say I wanted to get information on states located in only the Northeast. Those states would be states like Connecticut and they have a state region of one. I'm going to go back to my R-script and write a command that pulls in this data. This subset command would look like this. Here I'm passing the data frame states info to subset and I'm asking for it to retrieve any states that have a **state.region** equal to **1**.

```
6 subset(statesInfo, state.region == 1)
```

When I run this code I can see the output down below. We have Connecticut, Maine, and many others. Now, there is another way to subset this data frame. I didn't want to show you it, just so that way, you don't get confused later if you see it. It uses bracket notation, where we have the name of our *dataSet*, followed by two brackets. And we'll have a comma in between. This first spot is for the *rows* of our data set that we want to keep. And the second spot is for the *columns* that we want to keep.

```
> subset(statesInfo, state.region == 1)
                X state.abb state.area state.region population income
7     Connecticut        CT       5009            1       3100   5348
19          Maine        ME      33215            1       1058   3694
21 Massachusetts         MA       8257            1       5814   4755
29 New Hampshire         NH       9304            1        812   4281
30    New Jersey         NJ       7836            1       7333   5237
32      New York         NY      49576            1      18076   4903
38  Pennsylvania         PA      45333            1      11860   4449
39  Rhode Island         RI       1214            1        931   4558
45       Vermont         VT       9609            1        472   3907
```

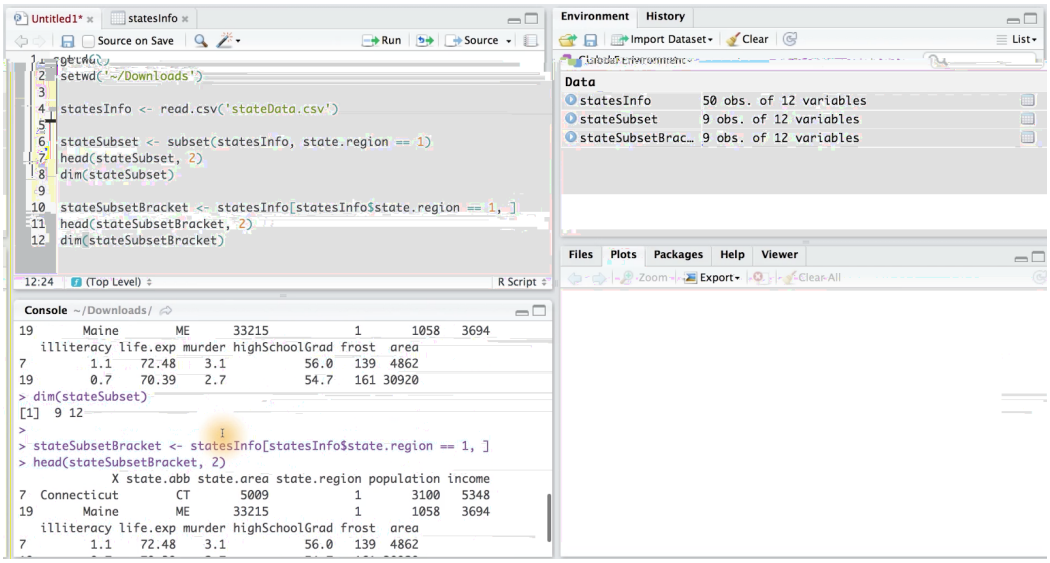
```
dataSet[ROWS,COLUMNS]
```

So if I want only the states in the Northeast, I would write this code. The name of the data set is states info and then I want the rows that have a state region equal to one. Now I can't just use state region here, I need to access the actual variable, so I have to put states info and the dollar sign. This gives me the actual variable value and I can see if it's equal to one. If it is equal to one, I want to return every single column in the

```
8  statesInfo[statesInfo$state.region == 1, ]
```

data frame. So for example, with Connecticut if it's state region is equal to one. I want to return every single column in this row.

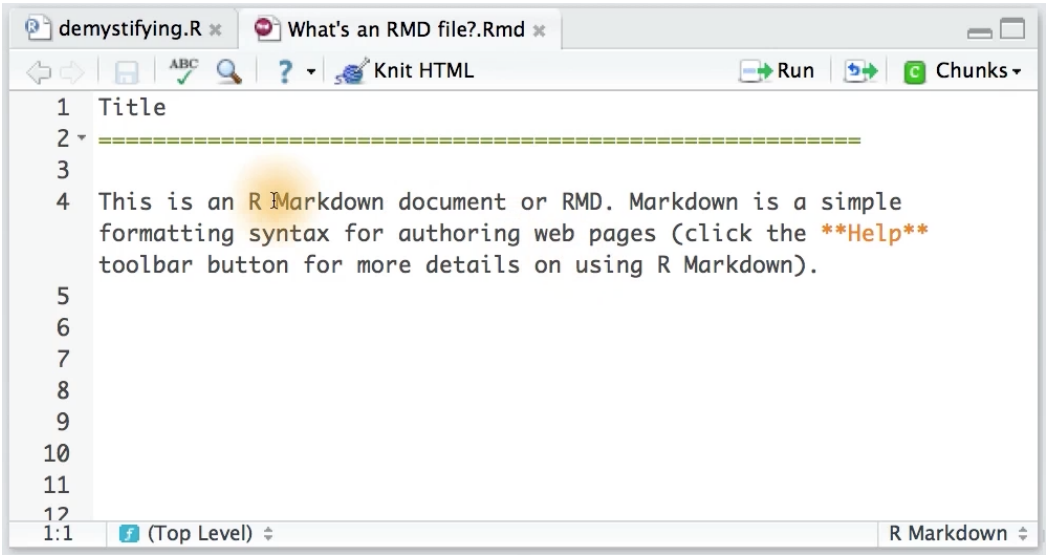
To return all of the columns, I'll just leave this blank. So this code searches for rows that have a state region equal to one. And then it takes all the columns in that row. And all of this will be sent to the console as a new data frame. Now it might not be so helpful to have this output just in the console. So we can save these subsets into new variables. I'm also going to include some functions to print out the first two rows of each data frame, and also their dimensions. Hopefully, I've convinced you that they're the same data set.



Now, I really want you to pay careful attention to the syntax in both of these examples. Throughout this course, we tend to make use of the subset command, but there might be instances where we use the other method. Just know that both methods produce the same result. Now, I recommend that you try subsetting this data frame for other regions of the country on your own. You could also try finding out which states have an illiteracy rate of 0.5%, or which states have high school graduation rates above 50%. Feel free to play around.

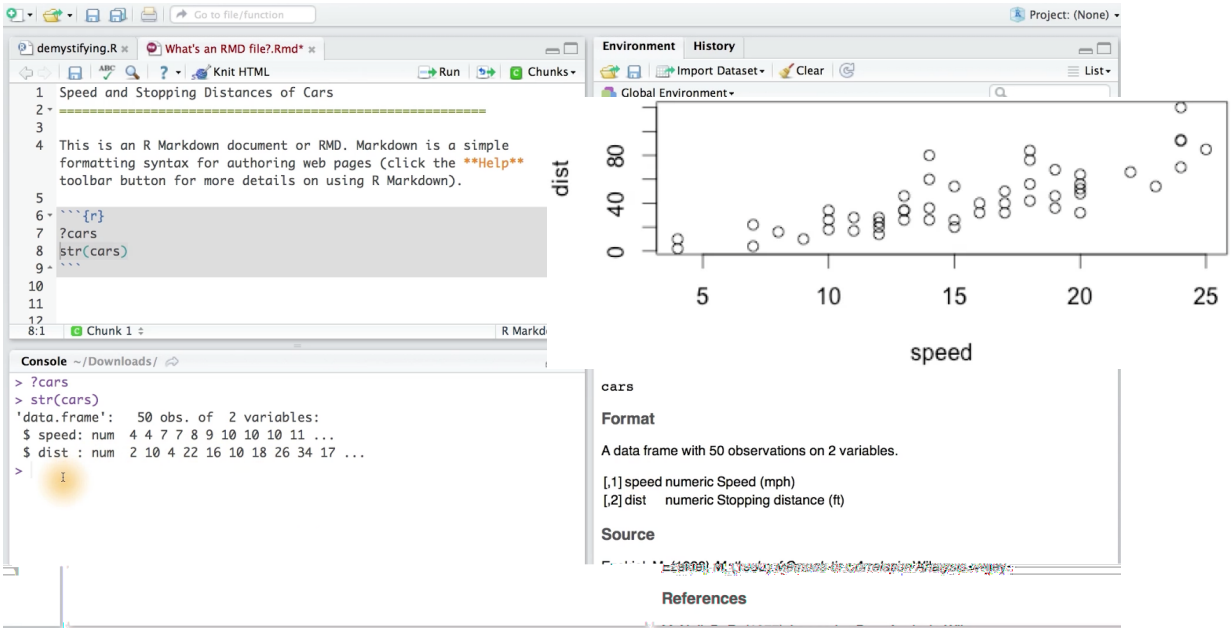
R Markdown Documents

Let's get some more practice working with data frames. You downloaded an R script earlier and saw how we can save our work and run an R code from it. For your next task, you're going to download an RMD file and run code in it. The file will look something like this. Notice how this file is slightly different from the file that you saw before. An R script can only have R code and comments. This file, however, the RMD file, allows us to do so much more. It's an R Markdown document, or RMD.



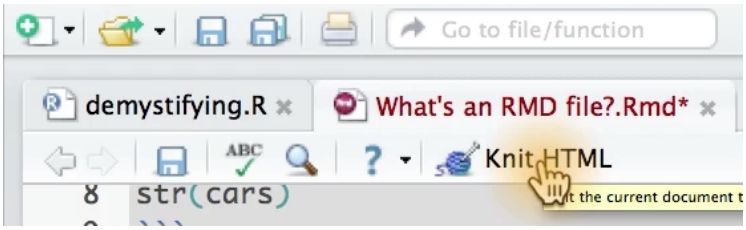
Now, you might be wondering what Markdown is. And basically, it's a simple formatting language that lets us author web pages. If you want to know all you can do with Markdown then go to Help and then go to Markdown Quick Reference. This would bring up the documentation for Markdown, and you can scroll through some of the examples in the formatting. We've also included a video that will go through Markdown.

So far this file only contains text that will be formatted using Markdown. Let's add some R code to this. We can do this by clicking on Chunks and then going to Insert Chunk. Now if you're friendly with the keyboard, you can use the shortcut Cmd+Option+I. There's many other shortcuts in here and you can see them here in this menu. Here I've added some code to see what this data set is. This is the cars data set that also comes with R, and it contains 50 observations of speeding and



stopping distances for cars in the 1920s. I'm going to add this text as the title to this document, so that way it's more descriptive of my file. I'll also run the **str** command on this data frame, so that way I can see what's inside of it. It looks like I have two variables, speed and distance, which are measured in miles per hour and feet. Not only can we run R code and get the output from these files, we can also embed images. This **plot** function comes with a base graphics package in R. But don't worry learning more about it. We'll be using the **ggplot2** package which is a powerful graphics language in the coming lessons.

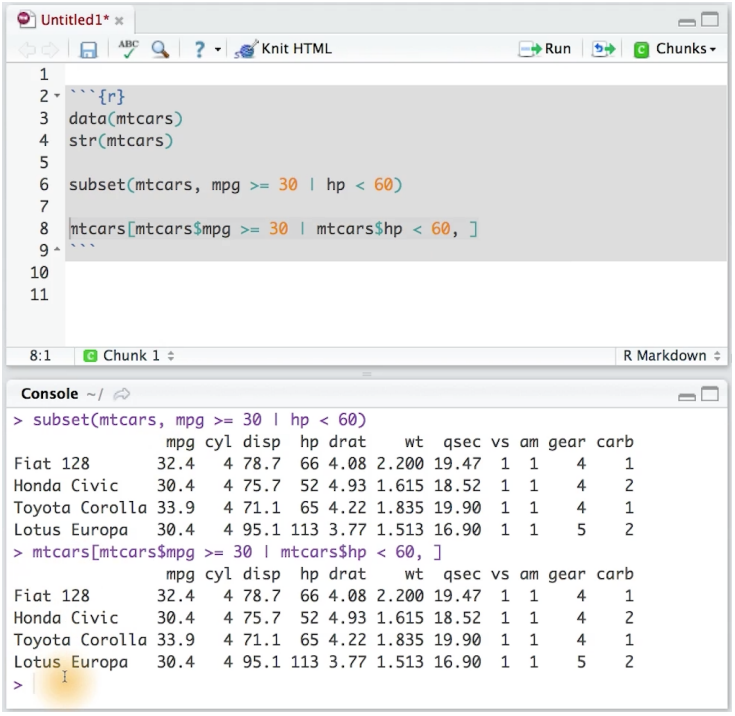
When I run this code - **plot(cars)** - I get a simple plot of speed versus distance. Now, what's really amazing about this document is that we can click the Knit HTML button. This will generate a web page that includes both content, as well as the output of any embedded R code. First, you'll need to install and load the package **knitr** in order to use the Knit HTML button. Run the commands to the right in RStudio console install and load **knitr**. This simple button allows us to easily share and publish our work. That's enough about RMD files. Let's have you work in one now. Download [the second RMD file] if you haven't already, and run and write code when you're prompted to do so. At the end of the file, you'll come across a question. I want you to answer that question for this next quiz. And then you can continue on with the rest of the lesson. Good luck and have fun.




```
install.packages('knitr', dependencies = T)
library(knitr)
```

Answer

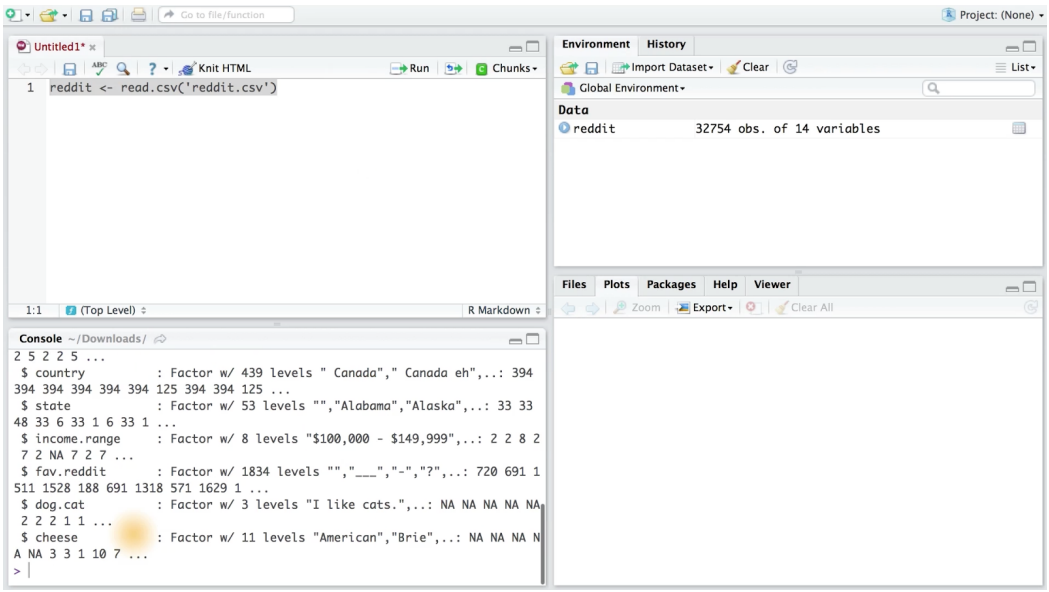
The cars that satisfy either this condition or this condition were the Fiat 128, the Honda Civic, the Lotus Europa, and the Toyota Corolla. Let's see how we can figure this out using "R". Your task was to determine which cars in the **mtcars** data set have MPG greater than or equal to 30 and an HP less than 60. To answer this question you needed to subset the data frame. I'll show you two ways of doing this. In the first method, I'll use the **subset** command on empty cars, and I want to get the cars which have an mpg greater than or equal to 30 or whose horse power is less than 60. That would be the Fiat, the Honda Civic, the Toyota Corolla, and the Lotus Europa. Using the bracket notation, the coats and tacks would look like this, and there's the same output.



Factor Variables

Now that you're familiar with the basic R commands, let's look at some more data. This time we'll be looking at data collected from a survey of Reddit users. Reddit's a social and entertainment website where users can post links and comments about trending news. This survey asks users about demographic information such as gender, age, nationality and employment status. It even asks users what type of cheese they would be. And whether they prefer dogs, cats, or turtles. Download the data set from the instructor notes and load it into R. Once you've done that, take a look at the data by using the **str(***data***)** function.

Now when I try to read in the file, sometimes I might get an error, and this is pretty common, so I would suggest looking at your current working directory to figure out the problem. Often times your directory isn't where your file is stored. Alright, so I've set my directory, and now I'm going to try this code up here again. And there we go,



there's our data. Running the **str(reddit)** command, we can see that there's lots of data here. Most of these variables have a type of factor.

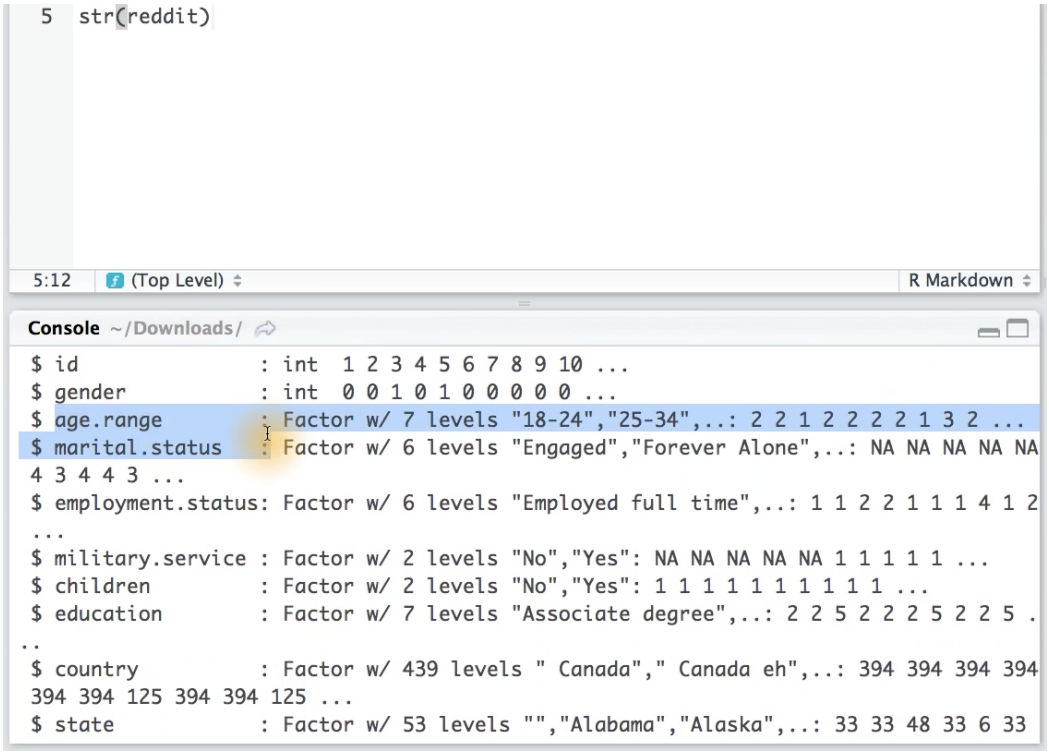
Now, a factor is a categorical variable that has different flavors or levels to it. An example of this would be employment status. This variable has many different levels such as employed full time or employed part time or not working. One thing we might be interested in is how many people are in each group of employment status. We can table that variable to see the number in each of these groups. Running this code - **table(reddit$employment.status)** - I can see the table.

```
> table(reddit$employment.status)

                   Employed full time                               Freelance
                                14814                                    1948
Not employed and not looking for work    Not employed, but looking for work
                                  682                                    2087
                              Retired                                 Student
                                   85                                   12987
```

We can also get these counts and other data points by running the **summary(reddit)** function on our data frame. I encourage you to check out the output of this for yourself. In addition to factor variables like employment status, the R programming language also has other data types like list and matrices, but we really won't be working with them in this course. We've included a reference for data types, so if you would like to learn more about those, check it out.

Ordered Factors

Let's look more closely to these factor variables. For now I want to draw your attention to the **age.range** variable right here. Notice that it says that we have a factor variable with seven different levels. We can examine the levels of a variable, by

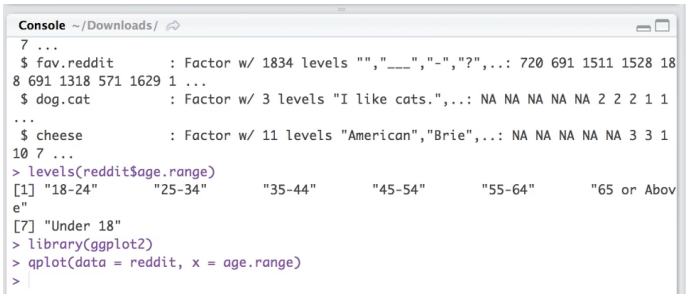


typing in the command **levels(***factor variable***)** and then putting in the variable to the argument. In the console we can see the seven levels of the **age.range** variable. Now,

```
> levels(reddit$age.range)
[1] "18-24"       "25-34"       "35-44"       "45-54"       "55-64"       "65 or Abov
e"
[7] "Under 18"
```

instead of creating a table of the **age.range** variable, let's create a plot that shows how many users are in each bin. That is, we want to figure out how many surveyed respondents are between the ages of 18 and 24, 25 and 34, and so on. I'm gonna create this plot using the **ggplot2** package, and the **qplot** function that comes with it.

Again, don't worry about understanding this code too much, we'll have practice with

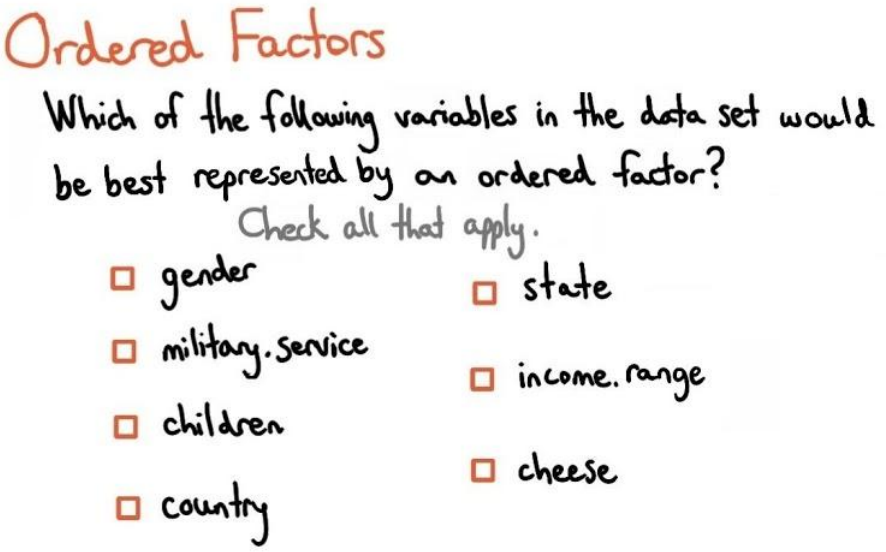


this in the next lesson. When I run this code, I get my plot over here. I want you to notice that the age groups appear to be in order. This is true for everyone except the survey takers who are under the age of 18. Now, it would be really helpful if the under 18 bar was really on the left of the 18-24 bar. That way we could make comparisons across the groups more easily. Now this is why we would want to have ordered factors.

The variable **age.range** just contains factors with seven levels, but these levels aren't arranged in any particular order. Sometimes you want to introduce order into our data set. So that way we can make more readable plots. So, knowing a little bit about ordered factors, let's see if you can answer this next question.

Quiz

If you haven't already done so, download the Reddit survey data and look at its structure. After you looked at the structure of the variables, try and answer this question. Which of these variables in the data set could also be converted to an ordered factor? Just like **age.range**. Check any of the variables that apply.

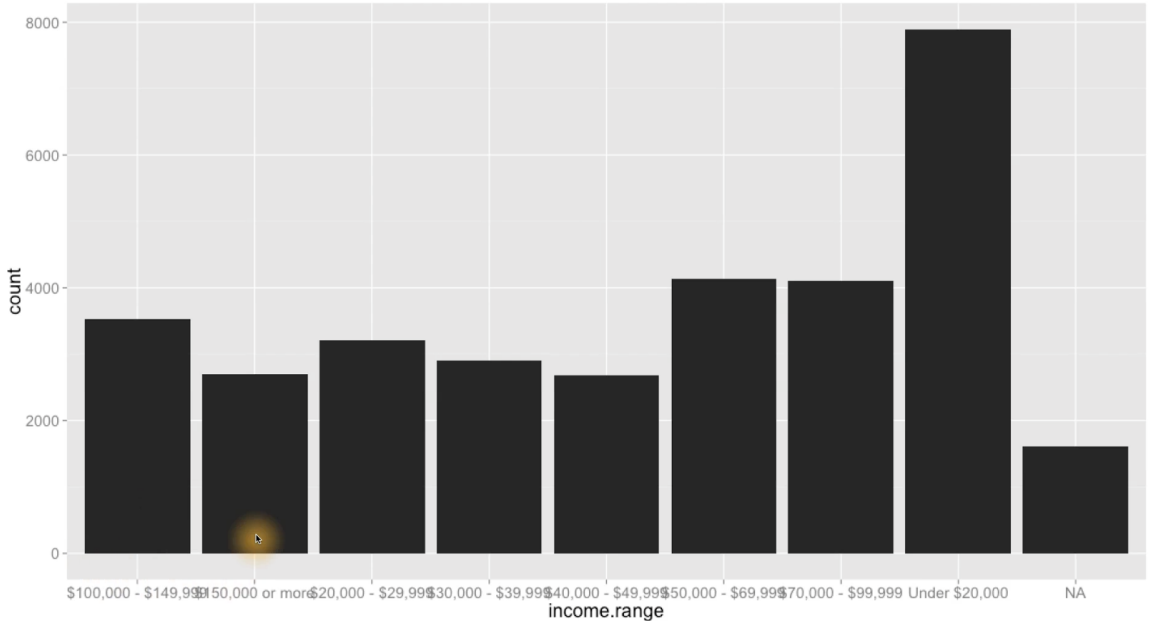


Answer

The answer here is just income level. Gender would not be considered an ordered factor, since it doesn't make sense to rank male and female groups. The military service and children variables are binary variables with yes or no values, so these really wouldn't make sense to order either. Using the same thinking, it doesn't really make sense to order the levels of country and state variables, since we wouldn't rank those either. We wouldn't really say one location is better than another. And finally, cheese falls into the same bucket as the country and state variables since they wouldn't necessarily be ranked either. The income levels actually have a natural order to them where we can group the buckets from lowest to highest, or from highest to lowest. That's what make it an ordered factor.

Setting Levels of Ordered Factors

So here's our problem. If we try to see the amount of users in each age group, it's not in order and it's hard to compare immediately. This problem is even more noticeable on a different plot. If we try to create the same plot for users in each income bracket, we see a very similar problem. And here it's much worse. These first two bins are for a 100,000 and over 150,000. And this last bin is for people who make under 20,000. Our eyes tend to read pages from left to right , so this graph is pretty hard to interpret. Or even better said, it'd be very hard to make comparisons naturally with our eye. We have to always scan down to the bottom to figure out what group we're in. So let's order the factors in our **age.range** variable.



Quiz

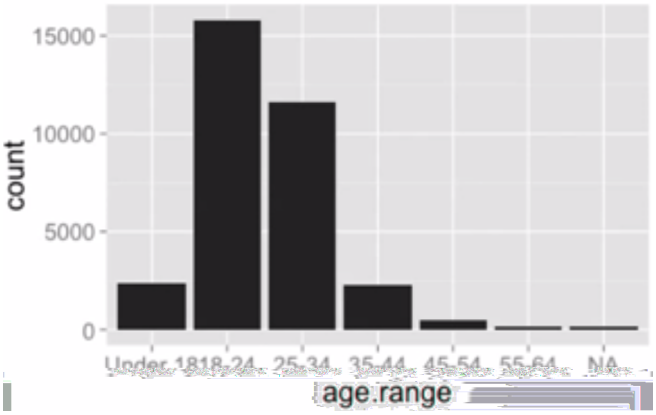
What I want you to do is to look up the documentation for the **factor** function. Or you can read through the example here. Once you're ready, try to write the code in order

to order the levels of the **age.range** variable. And just as a reminder, the level should take on the following values, 18-24, 25-34, 35-44, 45-54, 55-64, 65 and over, and under 18.

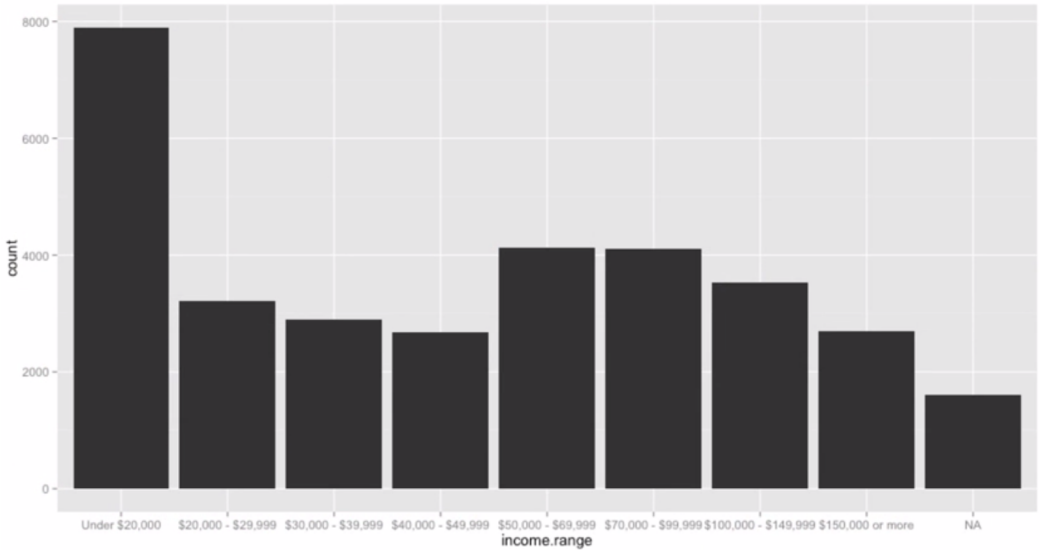
```
# Setting Levels of Ordered Factors Solution
reddit$age.range <- ordered(reddit$age.range, levels = c('Under 18', '18-24',
'25-34', '35-44', '45-54', '55-64', '65 of Above'))

# Alternate Solution
reddit$age.range <- factor(reddit$age.range, levels = c('Under 18', '18-24',
'25-34', '35-44', '45-54', '55-64', '65 of Above'), ordered = T)
```

Answer

For this programming assignment, you needed to have rearranged the levels of the **age.range** variable, so that way, under 18 appeared first. One way to do this is to use the **ordered** function on the variable. We'll use the ordered function on **age.range,** and then we'll set the levels. I can run all this code to set the levels, and then, I can make my plot again. Pretty nice, huh? Another way to achieve the same result would be to use this code. Here, we're using the factor function. We're taking our age.range variable, setting the levels, and then making the order be true since we want these ordered. I'll run this code. And then, I'll make sure that the graph is the same output. And yes, it is. Now, if you want another challenge, try reordering the levels for the income range variable, and see if you can produce this same plot.



Data Munging



It's important to note that many of the data sets that we've used so far in this course are what I would call tidy data sets. What I mean is that these data sets were

manipulated into a specified format of rows and columns. You can learn even more about tidy data here. This is what allowed us to so easily import these data sets into R. You should know that not all data sets are going to be so nice. Sometimes, you may be pulling data from different sources, like webpages, audio files, or even PDFs. Other times, you may need to reshape or rearrange your data into different formats. We're going to cover this later in lesson 5.

mtcars | reddit | statesInfo

32 observations of 11 variables

	row.names	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4

The important thing to know is that this is a necessary step prior to conducting EDA and it's called data munging. There are plenty of tools for doing this kind of work, and if you're interested in learning more about how to wrangle and adjust data, check out our data munging course. Now, this course EDA won't cover data munching, but techniques for doing so are vital for any data scientist.

Advice for Data Scientists

Before we wrap up this lesson, let's hear from Eytan Bakshy and Sean Taylor, who are data scientists at Facebook. They'll offer their advice for how to work with data.

Eytan: My advice to data scientists, or future data scientists is find data sets that you're interested in, and, and work with them, and play with the data. And just developed experience with playing with data.

Sean: Good data science comes from good questions, not from fancy techniques, or from, you know, having the right data. It comes from motivating your research with an idea that you care about, and that you think other people will care about. It's almost, you almost have to think about it like a journalist would. You have this audience and they're going to consume the work that you produce, and you want them to care about it, and be interested. And so, starting with the right questions, in particular, something that motivates you as a person is really the right starting point, not, like, you know, some fancy technique that you want to learn.

Congratulations

Congratulations on finishing lesson two. In this lesson you learned how to use R Studio and you learned about the basic commands in R. If you found something particularly helpful or if you have ways that we can improve the course, let us know by posting in the forum. In the next lesson, you'll learn how to visualize and summarize single variables within a data set. We hope to see you there.