Using R – Access and Data Entry

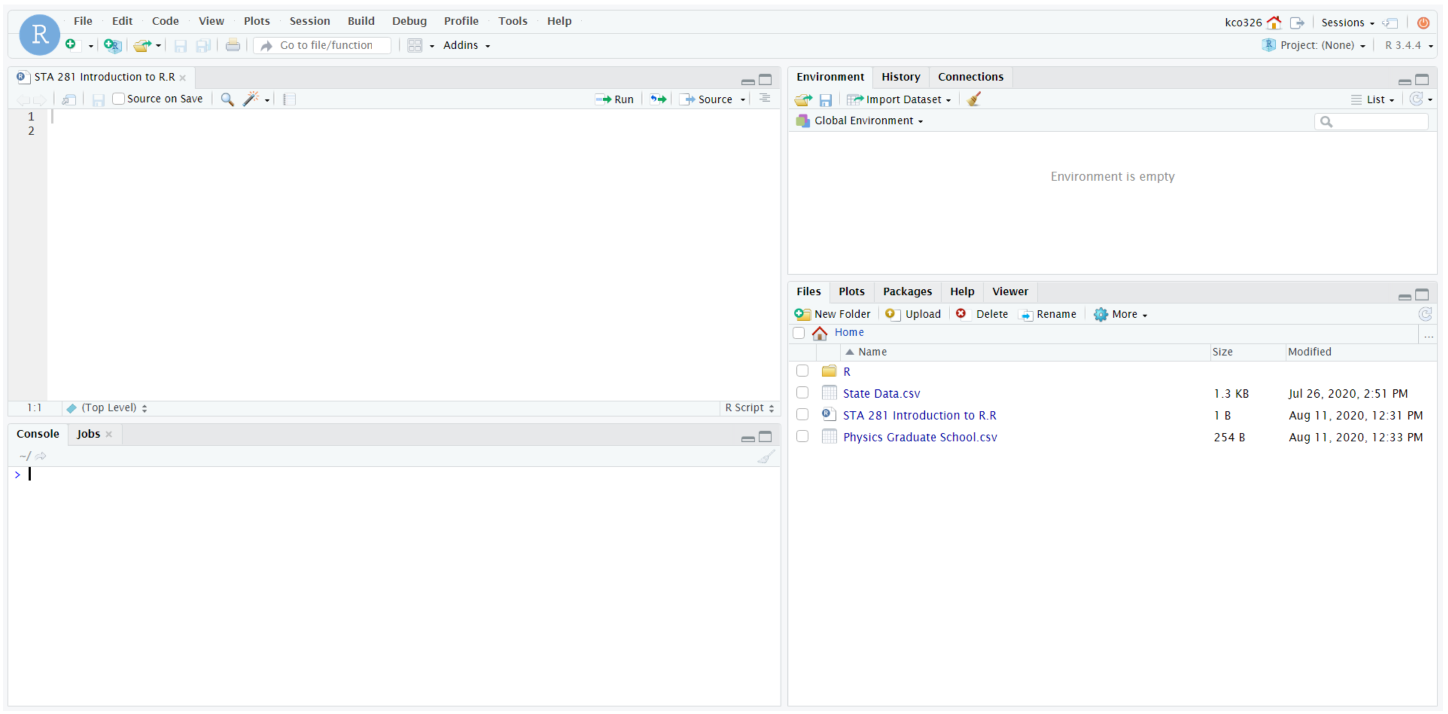
Accessing R and RStudio

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms. You can download R here: https://www.r-project.org/

RStudio is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace. You can download RStudio here: https://rstudio.com/products/rstudio/download/

If you would rather not download the software to your computer, we also have an RStudio server at UK. I have submitted each of your UK usernames and you should be able to log in with your UK credentials at this site: https://rstudio.as.uky.edu/ If you are unable to access the server, let me know.

When you log in, you should see something like this:



Environment is empty

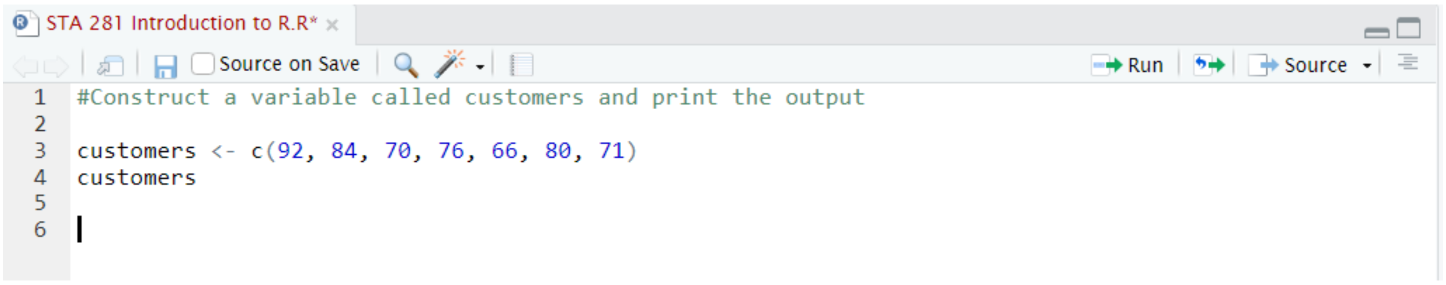
Name	Size	Modified
R		
State Data.csv	1.3 KB	Jul 26, 2020, 2:51 PM
STA 281 Introduction to R.R	1 B	Aug 11, 2020, 12:31 PM
Physics Graduate School.csv	254 B	Aug 11, 2020, 12:33 PM

Notice that there are four quadrants to RStudio. The upper left quadrant is where you will enter your code that can be saved and will be a permanent part of the file.   You can also enter code directly into the console (bottom left), but it will not be saved as part of your program. The bottom left quadrant gives output when a line or section is run. The upper right quadrant lists variables in the global environment and he bottom right quadrant shows uploaded files, graphical output, packages, and help. The split screen gives you a global picture of what is happening as you run your code.

Manual Data Entry

Data can be entered manually using the "c" function. "c" stands for "combine" – the "c" function combines its arguments into a single vector. If you want to apply another function to the vector in a separate command, you need to store the vector in R first. To do that, use the assignment operator "<-". You can call the vector whatever you want, but try to keep the name short and descriptive, and try to avoid using a name that is also the name of a commonly used built-in function (like "mean" or "hist").

For example, suppose that on a sample of seven days, the numbers of customers in a small restaurant were 92, 84, 70, 76, 66, 80, and 71. I'm going to store these values in a vector called "customers". I will start by entering the lines of code in the upper left quadrant:

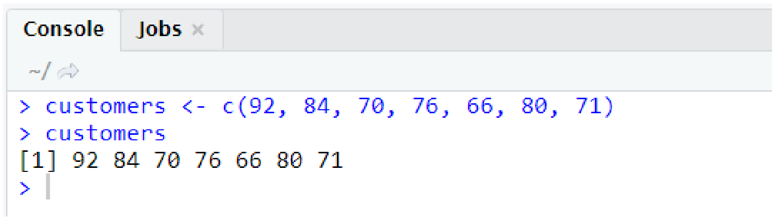


```
#Construct a variable called customers and print the output

customers <- c(92, 84, 70, 76, 66, 80, 71)
customers
```

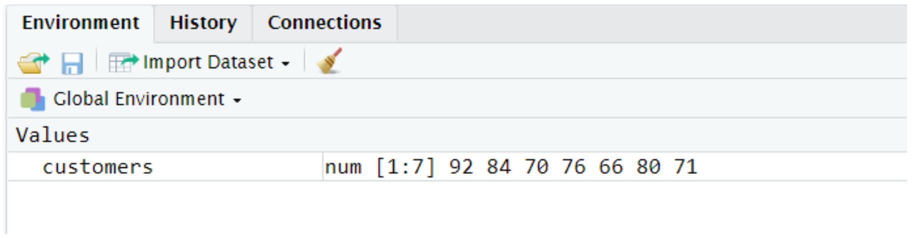
To run each line of code, the cursor just needs to somewhere on the line of code, while you press Run. If you highlight an entire section of code, you can run it all at once. You can document your code or make comments while programming as well. Anything following a # will turn green and will not be considered code by R.

When I run this code, I will get the following output in the console area:



```
> customers <- c(92, 84, 70, 76, 66, 80, 71)
> customers
[1] 92 84 70 76 66 80 71
>
```

You'll also notice that the variable "customers" shows up in the global environment on the top right side of the screen:



```
Values
  customers        num [1:7] 92 84 70 76 66 80 71
```

If you are interested in a particular value in the vector, you can obtain it by using square brackets to specify the position you want. Suppose you want the value in the second position of the vector:

```
#Find the second value in the vector customers

customers[2]
```

When I run this code, the output is:

```
> customers[2]
[1] 84
```

If you want values in multiple positions, you can use the “c” function to shorten the amount of typing you have to do. For example, suppose you want the values in the second, sixth, and seventh positions. You could obtain them one by one using three commands, or you could do it like this:

```r
#Find values in multiple positions

customers[c(2,6,7)]
```

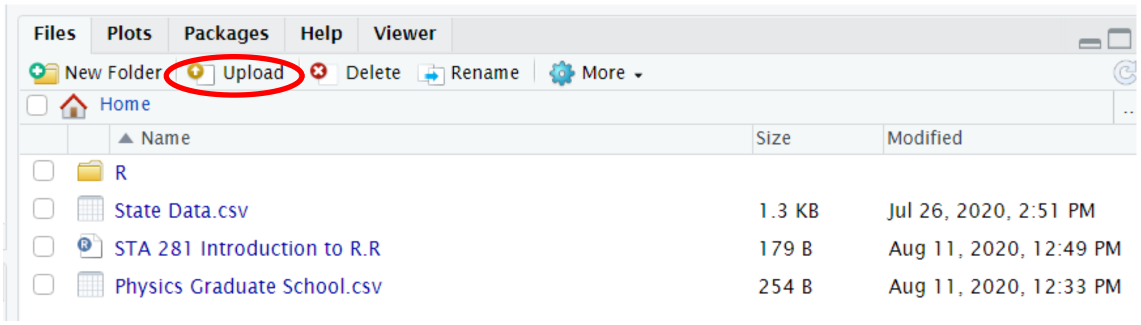
When I run this code, the output is:

```
> customers[c(2,6,7)]
[1] 84 80 71
```

Importing Data from a .csv File Using “read.csv”

Most of the time you will not be manually entering data but will be uploading a file of data. The most common format is a “csv” file.

The file “Physics Graduate School.csv” which is available in canvas contains the acceptance rates and TA stipends for ten graduate programs in physics. Once you have downloaded the file, you can upload it into your R environment using the upload function under the Files tab in the bottom lower quadrant. Once it is uploaded, it will show up as an available file:



Name	Size	Modified
R		
State Data.csv	1.3 KB	Jul 26, 2020, 2:51 PM
STA 281 Introduction to R.R	179 B	Aug 11, 2020, 12:49 PM
Physics Graduate School.csv	254 B	Aug 11, 2020, 12:33 PM

There are a couple of functions in R that you can use to import the data: read.table and read.csv. If you use read.table, you have to specify how the values are separated (for a .csv file, the values are separated with a comma, even though the file itself looks like a spreadsheet). If you use read.csv, it’s implied that the values are separated with a comma. That’s a little easier to work with, so that’s what we’re going to use first.

Using read.csv:

```
#Use read.csv to import the data

physics <- read.csv(file.choose(), header = T)
```

In the above code:

(1) I’m storing the physics graduate school data as “physics”. Just like before, the arrow is used for storing data.

(2) read.csv requires two arguments: the file location and name, and whether there is a header row or not.

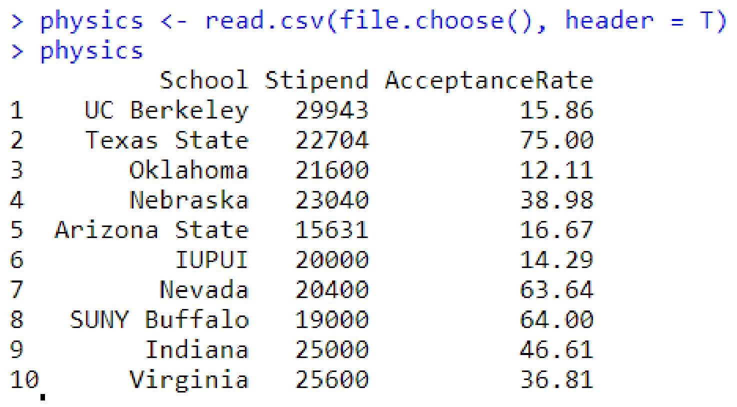
- (i) The easiest way to specify the file location and name is to use the function file.choose(). When you use file.choose(), you can select the .csv file interactively. It doesn’t look like much in an instructional document, but using file.choose() is almost always easier than typing or copying the file location.
- (ii) The data set has a header row, so header is set equal to T (for “true”).

Notes:

(a) Case sensitivity matters in the file name, as well as the column names (see next page).

(b) The numbers/columns in your .csv file should contain no formatting. For example, if you have monetary values, you don’t want to have any currency symbols in your .csv file. (In R, the dollar sign is an extraction operator – it can be used to extract things like specific columns or values from a frame/list, so having dollar signs in your .csv file will give you something far different than what you actually want.) Also, if you have values with more than three digits to the left of the decimal point, you don’t want to have any commas within them. (For example, if commas function as separators, “1,427.23” will be read as two values: “1” and “427.23”.)

Here is the output available in the console when I call the variable "physics""



```
> physics <- read.csv(file.choose(), header = T)
> physics
          School Stipend AcceptanceRate
1    UC Berkeley   29943          15.86
2    Texas State   22704          75.00
3       Oklahoma   21600          12.11
4       Nebraska   23040          38.98
5  Arizona State   15631          16.67
6          IUPUI   20000          14.29
7         Nevada   20400          63.64
8    SUNY Buffalo   19000          64.00
9        Indiana   25000          46.61
10      Virginia   25600          36.81
```

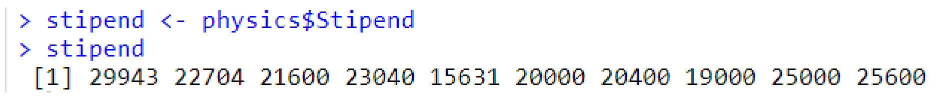
The data is now stored in R as a dataframe and you can see it in your global environment, but we can't do anything with it yet. Right now, it's stored as one big block, so we need to extract the relevant individual column(s) from it.

Storing Data from Individual Columns – Using the Header

If you want to store data from an individual column, the dollar sign can be used to specify the column you want. If you want to store the stipend column in "physics" separately, the R code should look something like this:

```
stipend <- physics$Stipend
```

Note: None of the functions used to store data in R naturally have output, which makes it hard to notice a mistake right away. However, you can always check by typing the name(s) you used to store the data (followed by "Enter"). For example, this is what you should have for the stipend data:



```
> stipend <- physics$Stipend
> stipend
 [1] 29943 22704 21600 23040 15631 20000 20400 19000 25000 25600
```

Storing Data from Individual Columns – Using Square Brackets

If you have a table, square brackets can be used to extract an individual element, row, or column. Below, [3,2] specifies the element in the third row and second column of the table called "physics":

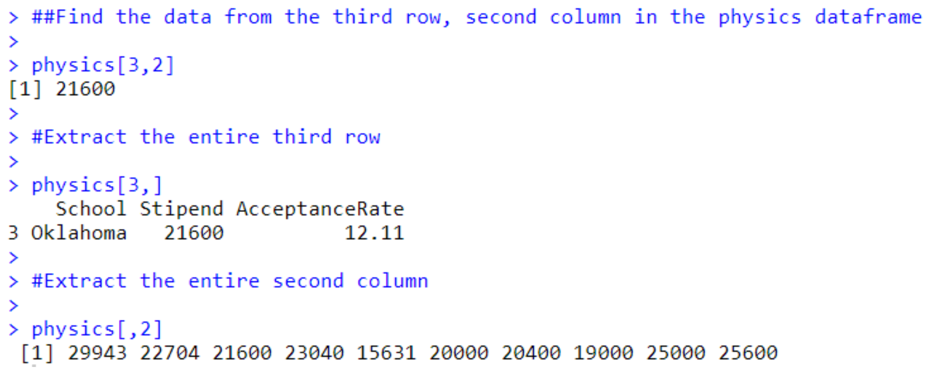
```
physics[3,2]
```

To extract a row, leave the column specification blank. To extract a column, leave the row specification blank.

```
physics[3,]
```

```
physics[,2]
```

Here is the console readout when these lines of code are run:



```
> ##Find the data from the third row, second column in the physics dataframe
>
> physics[3,2]
[1] 21600
>
> #Extract the entire third row
>
> physics[3,]
    School Stipend AcceptanceRate
3 Oklahoma   21600          12.11
>
> #Extract the entire second column
>
> physics[,2]
 [1] 29943 22704 21600 23040 15631 20000 20400 19000 25000 25600
```

Getting Rid of “NA” Values

If you import data with at least two columns of different length, R will add “NA” values to the shorter column(s), so their lengths match. In a lot of cases, you want to get rid of the “NA” values.

The file “Scottsbluff and Torrington Prices.csv” contains samples of prices of recently sold houses in Scottsbluff, NE and Torrington, WY. The samples are not the same size – there are 32 prices in the Scottsbluff sample, and 35 in the Torrington sample.

Download the data into canvas and upload it into your R environment. Use the read.csv function to import the prices into R and print the output:

```
scotts.torr <- read.csv(file.choose(), header = T)
scotts.torr
```

You’ll notice that the last couple of rows of output have “NA” values:

```
30      219900      209000
31      134500      325000
32      154900      163000
33          NA      355000
34          NA      205000
35          NA      267500
```

If you store the Scottsbluff data as a vector, this is what happens when you try to apply computational functions without removing the “NA” values:

```
> scottsbluff <- scotts.torr$Scottsbluff
> mean(scottsbluff)
[1] NA
```

The easiest way to get rid of “NA” values is probably to use the na.omit function. To omit “NA” values, the R command should look something like the one below. After the “NA” values have been omitted, computational functions will work.

```
> scottsbluff <- na.omit(scottsbluff)
> mean(scottsbluff)
[1] 256125
```

Importing Data from a .csv File Using “read.table”

The only difference between using read.csv and read.table is that you have to specify the separator if you use read.table. Use the “sep” argument to do that:

```
scotts.torr <- read.table(file.choose(), header = T, sep = ",")
scotts.torr
```