Importing Data Using R

This set of instructions will discuss setting R's working directory as well as a few different methods of importing data of different types. Note that there are other ways of importing data…these are just a few of the options available!

Setting the Working Directory (optional, but recommended!)

R is "pointed" at a particular directory in your computer. This directory, called the **working directory**, is the location R "looks at" when opening or saving files.

R will select a working directory by default. The command **`getwd()`** will show you this default working directory. For example, on my laptop, my default working directory is my "Documents" folder:

```
> getwd()
[1] "C:/Users/Claudia/Documents"
```

Why do we care about the working directory? If we have any data files that we wish to use in R, we want to be able to tell R where those data files are so that they can be imported. If we save these data files in our working directory, we can very easily call them into use in R (rather than having to type in their entire file name and location).

For example, say I have a data file called **example.csv.** If I have that file saved in my working directory (say, a folder on my desktop titled "Datasets"), all I need to do to import that data into R is type:

```
read.csv("example.csv")
```

and R will pull the file from the working directory. However, if I *don't* have that file saved in my working directory, I have to type in the file's full location (wherever that happens to be) to import the data:

```
read.csv("C:/Users/Claudia/Desktop/Datasets/example.csv")
```

Technically you don't need to set your working directory if you're okay with having to type the full file name/location whenever you want to import data, but I think it's a good idea to create a folder somewhere on your computer that you can set as your working directory. If you save all your data files in that folder, importing them into R becomes a lot easier!

Set Your Working Directory through R (PC Users)

There are two main ways you can set your working directory if you have a PC. You can use an R command or you can use a menu option. To use as an example for both of these, let's say I **create a folder on my desktop specifically for all of the data files for a class**. For this document, I'll create a folder called **Stat213Data.**

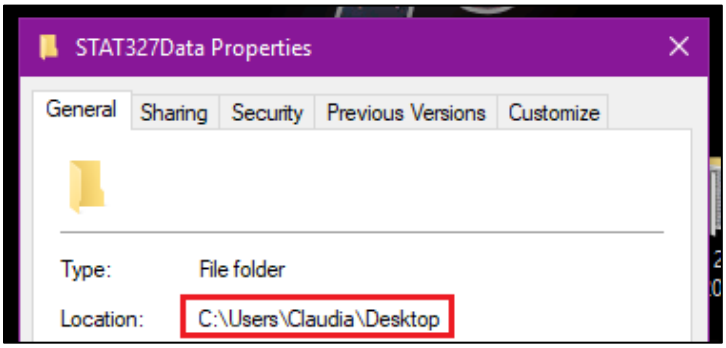
1. R Command

The command `setwd()` will set your working directory to whatever location you enter in the parentheses. Note that you have to put quote marks around the name of the location. If I want to set my working directory to that Stat213Data folder, I would type the following into the R console:

```
setwd("C:/Users/Claudia/Desktop/Stat213Data")
```

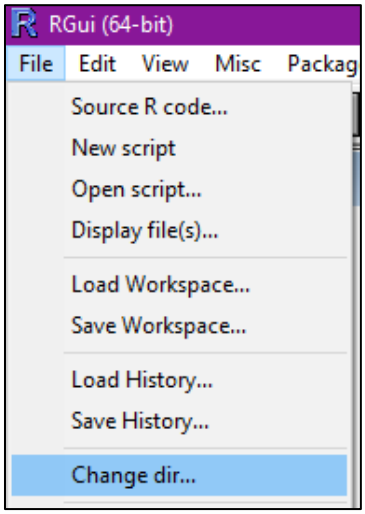
If I hit "enter" after this command and just get a new blank line, that's good! It means that R encountered no errors when trying to set the working directory.

If you're not sure of the exact location of a folder you'd like to use for your working directory, if you right-click the folder and select "Properties," you can see where the folder is located.

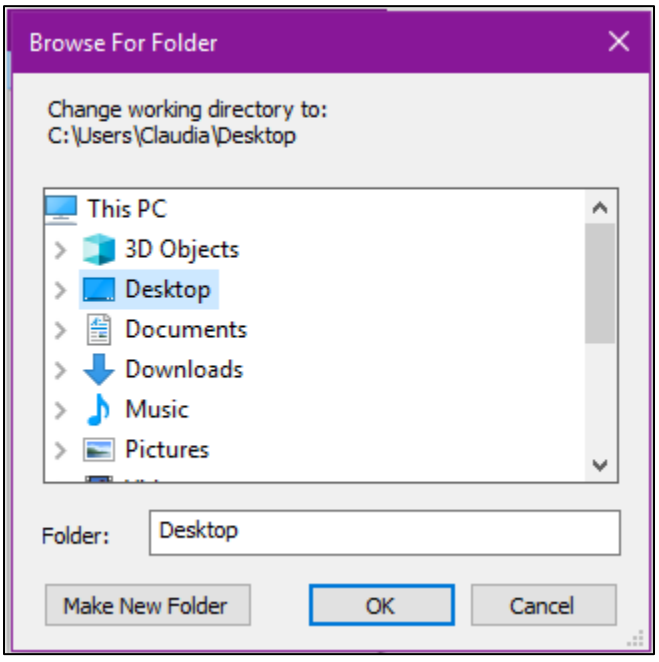


2. Menu Option

If you don't want to type in an R command, another method of setting up the working directory is by going to **File → Change dir…**



This will open a pop-up window where you can click through the locations on your computer to find (or create) the folder you want to be your working directory. Just select (or create) the folder you want and click "OK."



Now your working directory should be set!

Set Your Working Directory through R (Mac Users)

For Mac users, it sounds like using the command method to set the working directory can run into issues more frequently than it can on a PC. So for a Mac user, I would recommend using the menu to set your working directory.

Menu Option

On a Mac, you can change your working directory using the menus by going to **Misc → Change Working Directory.**

This will open a pop-up window where you can click through the locations on your computer to find the folder you want to be your working directory. Just select the folder you want and click "**OK**." Now your working directory should be set!

Importing a Data File

Importing a Data File Using R

Here are the R commands that can be used for various popular file types.

For a .csv file:

```
read.csv("filename.csv")
```

For a .txt file:

If your data file **starts with a row of column names**, you can tell R to read that first row of the data as the column names instead of as part of the data itself. This is done by including a `header = T` statement in the file import process.

```
read.table("filename.txt", header = T)
```

If you have data that does *not* start with a row of column names, just omit this `header = T` portion.

```
read.table("filename.txt")
```

Note that the read.csv command automatically checks for a row of column names.

What about Excel (.xlsx) files? Unfortunately, R on its own does not have a built-in function for reading .xlsx files. If you have data in an Excel file, the easiest thing to do is to open the file and save it as either a .csv file or a .txt file, then use one of the commands above to import the data.

Creating a Data File

Sometimes you may be given data that does not come in a downloadable file like a .txt or a .csv file. As an example, here's the data a problem on a WeBWorK assignment. It is just shown directly in the problem like so:

Winning Proportion - Y	Earned Run Average (ERA) - X
0.623457	3.13
0.512346	3.97
0.635802	3.68
0.604938	3.92
0.518519	4.00
0.580247	4.12
0.413580	4.29
0.407407	4.62
0.462963	3.89
0.450617	5.20
0.487654	4.36
0.456790	4.91
0.574047	3.75

There are a few ways of getting data like this into R: enter it manually or (if you can highlight the text) save it as a file that you can import using one of the methods above.

Manually Entering Data

I want to preface this section by saying that manually entering data is not typically the way to go, especially if you've got a good number of data points that you need to enter! It's very easy to make a typo that would then affect your whole dataset and it's also very tedious to do. However, there are some cases where you've just got a few numbers in your dataset and the easiest/quickest thing to do is just manually enter them.

For example, consider this dataset:

Bill	88.01	70.29	49.72	64.30	106.27	52.44
Tip	10.00	10.00	5.28	7.70	16.00	7.00

This is a pretty small dataset, so let's try entering into R by hand. The way we do this is by making vectors of each variable's observations.

To make a vector of values in R, we use the `c()` (combine) function. In this function, we enter all the values we want in that vector, each separated by a space.

Here's a vector of my "Bill" data (which I'll call "Bill," but you can name it whatever you want):

```
Bill <- c(88.01, 70.29, 49.72, 64.30, 106.27, 52.44)
```

And for the “Tip” data:

```
Tip <- c(10.00, 10.00, 5.28, 7.70, 16.00, 7.00)
```

Now if I type “Bill” or “Tip” in R, it knows what vectors I’m referring to and I can do other things with them (like compute the mean of the values, the standard deviation, do regression with them, etc.)

```
> Bill <- c(88.01, 70.29, 49.72, 64.30, 106.27, 52.44)
> Tip <- c(10.00, 10.00, 5.28, 7.70, 16.00, 7.00)
> Bill
[1]  88.01  70.29  49.72  64.30 106.27  52.44
> Tip
[1] 10.00 10.00  5.28  7.70 16.00  7.00
> mean(Bill)
[1] 71.83833
> sd(Tip)
[1] 3.738315
```

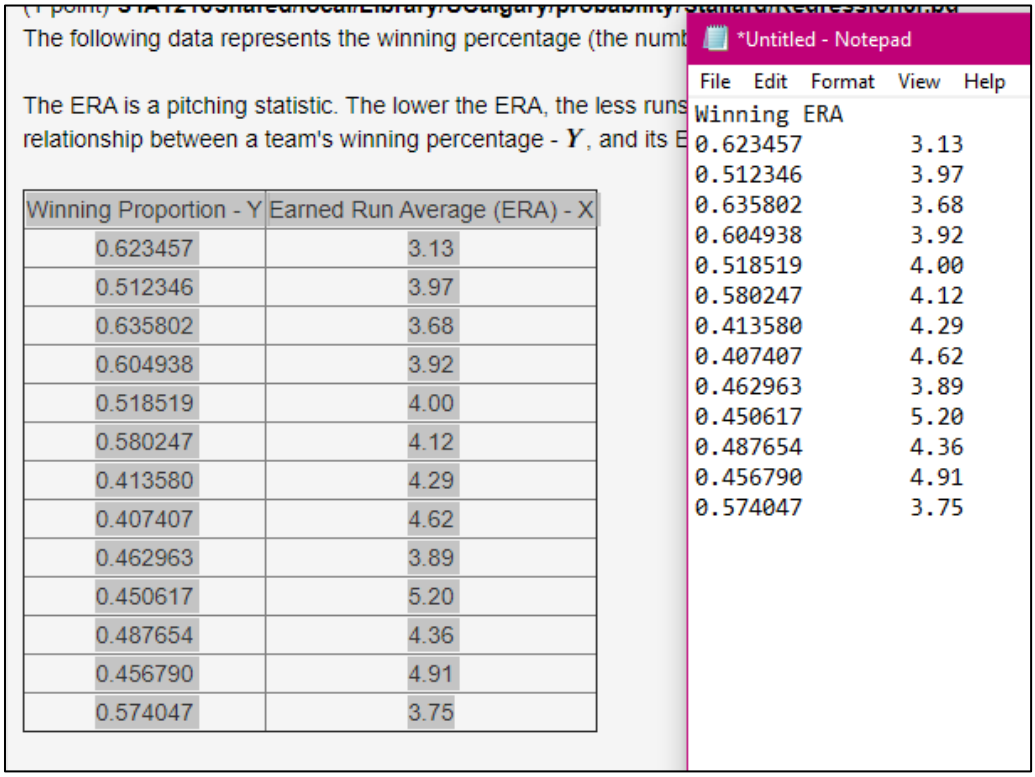
Note that if you type those vectors into an R or RStudio script file (recommended), you can save the script file so that you won’t have to re-type those vectors if you close R. You would just need to open the saved script file and run the lines with the vectors in them.

Saving Values as a Data File

Let's go back to this dataset:

Winning Proportion - Y	Earned Run Average (ERA) - X
0.623457	3.13
0.512346	3.97
0.635802	3.68
0.604938	3.92
0.518519	4.00
0.580247	4.12
0.413580	4.29
0.407407	4.62
0.462963	3.89
0.450617	5.20
0.487654	4.36
0.456790	4.91
0.574047	3.75

How can I get this data into R? Manually entering it would be very tedious! Probably the easiest way to get all of this information into R is to just select it all, copy it, and paste it into Notepad (or some equivalent program that saves things as .txt files).



Here I have just highlighted all the info in the table and pasted it into Notepad. Notice I also changed the names of the columns; this is because R sometimes doesn't have a good time with spaces in column headers and also because "Winning" and "ERA" are a little quicker to type when I need to refer to those two columns later.

Now I save that Notepad file in my working directory. This is data from question 6 of an assignment, so I'll name it "Q6."

To get it into R, all I need to type is `read.table("Q6", header = T)`.

```
> read.table("Q6.txt", header = T)
    Winning  ERA
1  0.623457 3.13
2  0.512346 3.97
3  0.635802 3.68
4  0.604938 3.92
5  0.518519 4.00
6  0.580247 4.12
7  0.413580 4.29
8  0.407407 4.62
9  0.462963 3.89
10 0.450617 5.20
11 0.487654 4.36
12 0.456790 4.91
13 0.574047 3.75
```

Assigning Names to Datasets

The commands above (`read.csv()`, `read.table()`, etc.) are used to import a dataset into R. But what if you want to do more with a particular dataset than just view it? One thing you might do is assign a name to that dataset so that you can access the information in that dataset more easily.

Assign a Name

Suppose I have downloaded a dataset named **students**. I want to import that dataset, but I also want to make it easy to refer to that dataset if I need to view it again or do anything else with it.

Suppose the **students** dataset is a .csv file. I would import it as follows (after setting the appropriate working directory):

```
read.csv("students.csv")
```

If I press "enter" after typing this command, the dataset will be displayed in the R console window.

Next, I'm going to assign a name to the dataset. This will allow me to refer back to the dataset for any other things that I might want to do (such as calculate means, standard deviations, perform a regression analysis, etc.).

I like to make my names easy to remember and easy to type, and if I'm just playing around with a dataset, the name I commonly choose for the dataset is just **x**. You can choose whatever name you like for a given dataset (as long as that name does not contain spaces), but if I want to call this dataset **x**, I type the following:

```
x <- read.csv("students.csv")
```

If I press "enter" after this, the data will not be automatically displayed. But if I just type the dataset name and press "enter," then I can view the data.

```
> x <- read.csv("students.csv")
> x
  X heights weights
1 1      73     195
2 2      69     135
3 3      70     145
4 4      72     170
5 5      73     172
6 6      69     168
7 7      68     155
8 8      71     185
```

Attach a Dataset

One thing that you should try to make an "automatic" next step after assigning a name to a dataset is to **attach the dataset**. Attaching basically makes R read directly from the dataset that's being attached. If you have column names in a dataset, attaching that dataset allows you to simply type the name of the column of interest in order to access it (and by "accessing" it, I mean to view it, do calculations on it, or graph it).

To attach a named dataset, just use the `attach()` command. For my **x** dataset:

```
attach(x)
```

will attach the dataset and allow me to reference a column name when wanting to access a specific column.

For example, I've just attached **x**, which is what I named my **students** data file. Now suppose I want to compute the mean height. Notice that **heights** is the name of the relevant column. So to get this mean, all I need to type is:

```
mean(heights)
```

Notice the difference in the results below. The first is what happens if I don't attach my dataset **x**. The second is what happens when I do attach my dataset **x**.

The dataset **x** is not attached:

```
> mean(heights)
Error in mean(heights) : object 'heights' not found
```

The dataset **x** is attached:

```
> attach(x)
> mean(heights)
[1] 70.11628
```

You of course do not *need* to attach a dataset in order to refer to specific columns. If you want to refer to a column called **col1** in a dataset you've named **x** and **x** is not attached, you would do so by typing the name of the dataset, then a dollar sign, then the name of the column: **`x$col1`**

```
> x$heights
 [1] 73 69 70 72 73 69 68 71 71 68
[26] 73 68 70 72 70 67 67 71 72 73
```

```
> mean(x$heights)
[1] 70.11628
```

Built-In R Datasets

R has a large variety of datasets that are "built in" with the program, meaning that you can use them without having to import anything. Some courses may make use of some of these built-in datasets over the course of the semester, so it is worth knowing how to access them.

Built-in R datasets already come with their own names. That means that if you type in the name of one of these datasets, R will automatically recognize it. As mentioned above, there are a lot of these types of datasets, but let's use one of them for the sake of demonstration.

The **`iris`** dataset in R contains four different measurements on 150 different iris plants, 50 each from three varieties. If I want to view this dataset, I just need to type **`iris`** into my console.

```
> iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
6            5.4         3.9          1.7         0.4     setosa
7            4.6         3.4          1.4         0.3     setosa
8            5.0         3.4          1.5         0.2     setosa
```

(Note that you will automatically be viewing the very end of the dataset; to see the top of it as in the screenshot above, just scroll up in the R console.)

As is the case with an imported dataset, it's easiest to work with a built-in dataset like this by first attaching it. That way R will recognize the column names. For example, if I want to find the average of the "Sepal Length" measurement, I can attach the iris dataset and then just use the **`mean()`** function on that column name.

```
> attach(iris)
> mean(Sepal.Length)
[1] 5.843333
```