Overview of R based

<http://cyclismo.org/tutorial/R/>

Intro to R, computing course

<https://www.codeschool.com/courses/try-r>

Advanced functions:

<http://stackoverflow.com/>

Download R Studio:

<http://www.rstudio.com/ide/download/>

R is a very powerful and useful computing language that will make your life much easier here at Crowdflower. However R can be very particular and comes equipped with some default settings that are not conducive to the type of spreadsheet manipulation you will be doing during your day to day activities. This can (at times) cause some very frustrating experiences and even deter people from using r all together. However if you follow some simple rules outlined in this manual, you can avoid many of the pitfalls that plague first time users.

**Getting Started:**

When writing a new r script the first thing you need to do is set the work directory. Make sure you have your file or files saved in the same place. Use the following function to set the directory.

setwd(“~/Desktop/”)

Make sure you surround the directory in quotes.

**Reading in files:**

There are many ways to read in a file (vector, factor, matrix,etc). R is unique in that it has a built in object type to read in spreadsheets called a dataframe. To read a file in as a dataframe simply use the ‘read.csv’ function:

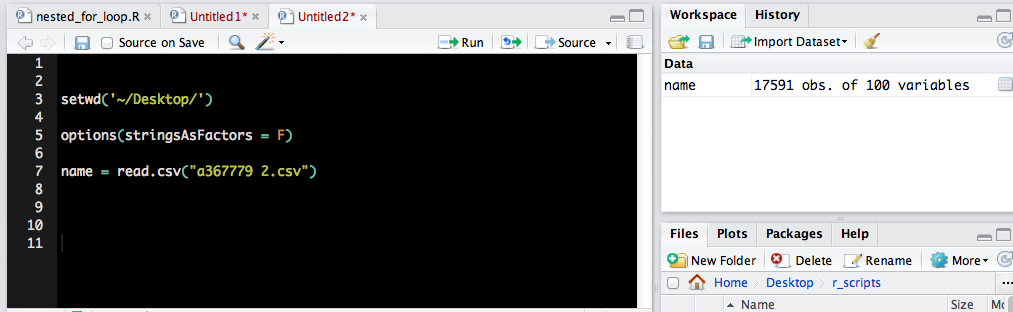
name = read.csv(“test.csv”)

Make sure you set a variable name for your file. R will not be able to read in the file as an object unless it has a name.

One of the pesky default settings mentioned earlier is that in some cases, r will read convert strings into another object type called factors. This can cause certain functions to operate differently than expecting resulting in a collision between your forehead and a desk. To avoid this get in the habit of **always** running the following function:

**options(stringsAsFactors = FALSE)**

So far you should have something like this.



Important to Know:

Some other pesky default settings in R that you need to be aware of.

* R will fill blank spaces with “NA”
* R adds an extra column numbering rows
* R will put an ‘X’ in front columns starting with a special character

**Removing Exxes**

Instead of a column being labeled “\_unit\_id” , R will output the column name as “X\_unit\_id”. This will keep you from uploading the file into a job correctly. To account for this run this simple function when preparing to write a csv:

name = names(**name\_of\_df**)

new\_names = gsub(“X\_”, “\_”, name)

colnames(**name\_of\_df**) = new\_names

**Remove NA’s and extra column**

To keep R from adding some unnecessary (and potentially job breaking) data into your csv append the following commands to write.csv function.

Default write function:

write.csv(name of dataframe, what you want the file to be called in quotes)

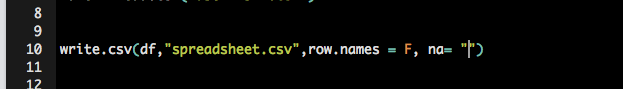
Remove extra column:

row.names = F

To keep ‘NA’s” from populating blank cells, add the line:

na= “ ”

Thus the final write.csv function should look like this:



**Basic Syntax**

$ specify the column you want to look at in a dataframe

[ , ] Brackets are used to subset or slice a data frame into smaller parts based off provided criteria. A good way to remember them is to thnk of them as “**which**” parts of the spreadsheet you want to look at. When using brackets in a data frame, you **must** specify by columns or rows (x or y axis) . This is why you will often find a comma followed by an empty space in a bracket pair. Ex:

[rows, ] = divide by rows

[ , column ] = divide by columns

[row , column] = divide by both rows and columns.

In most cases you will be divided spreadsheets by rows, so make sure to include a comma files by an empty space when using this command.

%in% intersect between dataframes

**Basic Functions:**

**Count Unique Rows**

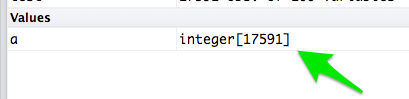
Count unique occurrences of a data type in a column.

The ‘unique’ function is great if you want to see if you have any duplicates in your data, how many “yes” or “no”, answers, etc.

Dummy = unique( [name of df] $ [name of column you want to check]

Macintosh HD:Users:bhazell:Desktop:Screen Shot 2014-01-15 at 12.24.52 PM.png

For this instance, I am looking for the amount of unique unit ids. I can see from the Workspace that I have 17591 unique unit ids.



To see which id’s are duplicates, run the ‘duplicate’ function.

duplicates = df [ duplicated (df $ column\_name ) == TRUE , ]

This function checks for the number of occurrences that are true, meaning they are duplicates. . The result is a new dataframe, labled “duplicates” that includes only columns with duplicate unit ids.

****

**Removing duplicates:**

At this point you have two dataframes.

Report = the entire dataframe

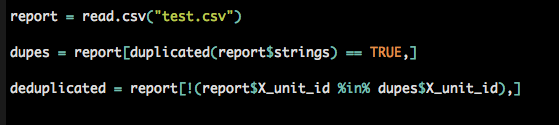
dupes = The duplicate versions of unit ids

To dedupe your original Report file, you simply want to tell R that you only want the data in the “Report” dataframe, that is not included in the “dupes” data frame. To do this we need to use the **%in%** (intersect) command.

Deduplicate = df1 [ ! (df1 $ column\_name %in% df2 $ column\_name) , ]

The (! ) means “not”, which means we want the data in df 1 that is **NOT** included in df 2.

Ex:



**Identifying which files Intersect based off unique identifier**

This is extremely helpful in tracking units, for example identifying which units have already run through a job an which ones haven’t using two different reports. For this we need to use the [ ]’s in combination with %in% .

For this example we are going to use to agg reports from Job 1 and Job 2. To see which units from the first job, that intersect with the second use the following line.

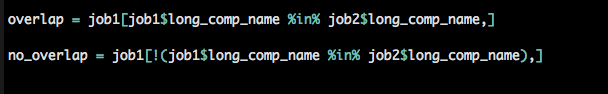
overlap = df 1 [ df1 $ column\_name %in% df2 $ column name , ]

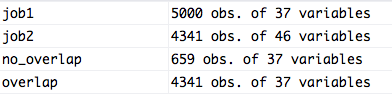
Overlap includes all units that are in df 1, that are **also** in df 2

Inversely, if you wanted to identify units that are in Job 1 and **not**  included in job 2 you can wrap the previous line of text in a not ( ! ) function.

no\_overlap = df 1 [ ! (df1 $ column\_name %in% df2 $ column name ) , ]

Ex:





If you would like to see which units from job 2 intersect with job 1, simple inverse the line of code.

Macintosh HD:Users:bhazell:Desktop:Screen Shot 2014-01-15 at 2.52.11 PM.png

Remember:

* The order of the data frames matter
* Always include a comma between brackets to signify the x and y axis
* Make sure you are using a **unique** identifier. It is always good practice to check for duplications before running this line of code as this will affect your output.

**Combining Data Frames**

To combine two data frames into one spreadsheet, use the “bind” funtions.

rbind = combine by rows

cbind = combine by columns

Example:

new\_output = rbind(job1, job2)

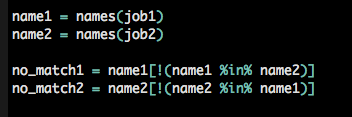
This is much faster than copying and pasting. Also, you don’t need to worry about order, R will automatically line up the right columns.

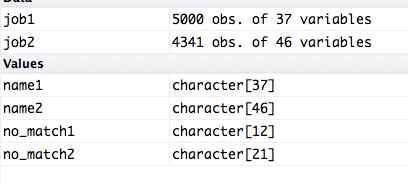
The basic r bind function will only work if you have the **same** amount of columns in each dataframe. If you try to combine two files with a different amount of data frames, R will throw an error.

To get around this there are two options.

**1. Identify which columns don’t match and delete them manually.**

An easy way to do this is to use the ‘names’ function in conjunction with the [ ] ‘s

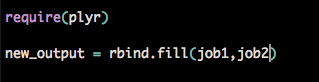




Note: Because the ‘names’ function creates a vecor, not a dataframe, you do not need to include a comma in the brackets.

**2. Force files to merge**

You can do this by using the rbind.fill function that is located in the plyr package.



The “rbind.fill” function will generate NA’s for columns that don’t match.

Ex:

