

# Data Cleaning - How to remove outliers & duplicates

Begin with reading in your data set... we'll use an example data set about schools.

```
library(formhub)
my_data <- formhubRead("~/Downloads/MyDataSet.csv",
  "~/Downloads/MyForm.json")
```

followed by selecting a variable that you want to do outlier work on. Let's look at the total amount of female pupils per school for this particular data set, labeled as `num_students_total_gender.num_students_female`.

```
library(ggplot2) #install.packages('ggplot2') may need to be run if you
don't have the package
qplot(data = my_data, x = num_students_total_gender.num_students_female) +
ylab("Number of Schools")
```

A quick eye-balling of the plot tells us that there are a couple of female student outliers that are quite high - as indicated by the extension of x-axis to 5000. Zooming in our plot may help look at the distribution better:

```
qplot(data = my_data, x = num_students_total_gender.num_students_female) +
ylab("Number of Schools") +
xlim(c(0, 1500))
```

## Round 1: na.strings

There is a weird-looking spike at 1000. If we zoom in, the problem looks to be right around 1000.

```
qplot(data = my_data, x = num_students_total_gender.num_students_female) +
ylab("Number of Schools") +
xlim(c(900, 1100))
```

```
qplot(data = my_data, x = num_students_total_gender.num_students_female) +
ylab("Number of Schools") +
xlim(c(990, 1010))
```

Oh, looks like the spike is of the value "999", which (in its negative version) is often used as a "Do Not Know" type of value in surveys. R lets us deal with individual values like this by specifying an `na.strings` parameter when

reading in csvs; this is exposed in the `formhubRead` function. So we can get rid of this value by re-reading our dataset while providing the `na.strings` parameter:

```
my_data <- formhubRead("~/Downloads/MyDataSet.csv",
  "~/Downloads/MyForm.json",
  na.strings = c("999"))
qplot(data = my_data, x = num_students_total_gender.num_students_female) +
  ylab("Number of Schools") +
  xlim(c(0, 1500))
```

Phew, no weird spike near 1000! If you are using values such as “-999”, “999” or something else for your “NA” values, it is good practice to include them in your `na.strings` when you call `formhubRead` or `formhubDownload`.

## Round 2: outlier cut-offs

However, our super-high outlier is still present at the dataset. At this zoom level, we that the vast majority of schools have less than 500 female pupils. For the sake of crudely setting our outlier paramaters, let's say that any facility reporting to have over 1000 female pupils will be counted as an outlier. The simple way to take this outlier out in R would be say something like `my_data$num_students_total_gender.num_students_female <- ifelse(mydata$num_students_total_gender.num_students_female > 1000, NA, my_data$num_students_total_gender.num_students_female)`. However, since besides being verbose, this method is also quite slow, we have written the following `outlierReplace` function. It takes a dataframe, a vector of columns (or a single column), a vector of rows (or a single row), and the new value to set to it (which we'll default to NA). This function makes it easy to write outlier-replacement commands, which you'll see below. You should feel free to copy this into your R scripts to do outlier replacements yourselves, but do note that the `outlierReplace` function will replace data in your dataframe directly.

```
# install.packages('data.table') may need to be run if you don't have the
# package
library(data.table)
outlierReplace = function(dataframe, cols, rows, newValue = NA) {
  if (any(rows)) {
    set(dataframe, rows, cols, newValue)
  }
}
```

Now, we will call `outlierReplace` on our dataset, where we'll replace all values in the column `num_students_total_gender.num_students_female`, for all rows in which the value is > 1000, with NA. Afterwards, we'll plot the graph without adjusting the x-axis, and see that the extreme value has been removed.

```
outlierReplace(my_data, "num_students_total_gender.num_students_female",
  which(my_data$num_students_total_gender.num_students_female >
    1000), NA)

qplot(data = my_data, x = num_students_total_gender.num_students_female) +
  ylab("Number of Schools")
```

It is also possible to use the `outlierReplace` function to change the value of more than one data point. Using the same outlier limit of 1000 for instance, we can change both the number of female pupils and the total number of pupils to NA like so:

```
outlierReplace(my_data, c("num_students_total_gender.num_students_female",  
"num_students_total_gender.num_students_total"),  
which(my_data$num_students_total_gender.num_students_female > 1000), NA)
```

Finally, instead of changing outliers to NA, we could make them equal to a maximal number. To do this, and show you a clear results, we'll take all observations with more than 500 female students, and cap them at 500.

```
outlierReplace(my_data, "num_students_total_gender.num_students_female",  
which(my_data$num_students_total_gender.num_students_female >  
500), 500)  
qplot(data = my_data, x = num_students_total_gender.num_students_female) +  
ylab("Number of Schools")
```

Note that the data has a much narrower range, and a spike at 500 now.

## Caveat: replace data with extreme care

Okay, so that cap of 500 was just a quick demo, lets undo that. But how? We can't simply replace the value with 500 with something else, because it could have been anything 500 or above. Replacing data is hard to undo easily, so be careful with functions like `outlierReplace`! In order to undo, we will have to re-read our dataset, and re-perform all the actions before the replace. Thankfully, however, we haven't saved our data, and there is only one thing we did before the replace, which is easy to re-create:

```
my_data <- formhubRead("~/Downloads/MyDataSet.csv",  
"~/Downloads/MyForm.json")  
outlierReplace(my_data, c("num_students_total_gender.num_students_female",  
"num_students_total_gender.num_students_total"),  
which(my_data$num_students_total_gender.num_students_female > 1000, NA))
```

## Saving your dataset

Don't forget to save your new data set!

There are two ways to do the save. If you want all the form information preserved (and maybe the ability to run functions like `replaceHeaderNamesWithLabels` in the future, you can save the `formhubData` object as is, in an `.rds` file. Only R will be able to read this file. If you need a widely usable file, then use `data.frame`, and save the data frame, for example as a `csv`.

Option 1:

```
saveRDS(my_data, file = "~/Desktop/MyFormhubData.rds")  
# later, you can do my_data <- loadRDS('~/Desktop/MyFormhubData.rds') to  
# re-load this same object
```

Option 2:

```
write.csv(data.frame(my_data), file = "~/Desktop/MyFormhubData.csv")
```