## **Dirty Data Project Feedback: Malcolm**

## **Positives**:

Overall:

This was an excellent submission full of:

* Best coding practices
* Good directory structure
* Comprehensive cleaning
* Working according to tidy data principles

It’s clear you got to know the data and you used this information to fuel your cleaning and wrangling operations to prepare the data for analysis. Your documentation was good and included the relevant sections we wanted to see and you answered all of the questions well. Well done!

Cleaning Script:

I liked that you had separate cleaning scripts for each dataset and then another for combining the data. This means someone running your code doesn’t have to run everything to view just the clean 2016 data for example.

You made good use of dplyr, pipes, and followed tidy data principles. You also indented well, employed white space appropriately and used meaningful variable names. Using relative filepaths was good and helped make your code reproducible - I had no issues running any of your scripts or analysis files.

Your cleaning steps were good and well thought out. You clearly explored the data well, made good observations and then carried out steps accordingly.

Cleaning the country column was comprehensive.

Analysis:

Good to see a list of the assumptions you made. All reasonable.

Good documentation of cleaning process.

Answered all of the analysis questions well, making good use of dplyr and pipes. Correct use of the here package and relative file paths was good for reproducibility. Loading libraries and formatting code was done according to standard practice.

## **Potential Improvements:**

Overall:

Folder structure was good. The root folder has a name with spaces - Task 4 - Halloween Candy Data. We recommend using snake case as it makes it easier when referencing file paths. E.g. task4\_halloween\_candy\_data.

Included in your repo was a working\_out folder. This is fine during development but for a final submission this should be removed.

More comments! It’s important to comment your code as well as provide informative documentation. Comments allow others (and your future self) to understand what decisions have been made and why.

Cleaning Script:

Referencing columns by numerical index is fine but referencing by name is preferred - easier to read and debug. If you’re going to reference by numerical index, leave a comment to show what it’s in those columns you’re pivoting/not pivoting. Explain why you’ve taken the steps you have.

Again, a comment as to why you’ve created new columns and filled them with NAs (gender, and country). I might know because I’m familiar with this project. But another user, new to the project won’t understand why this step was taken.

Cleaning of the country columns was comprehensive but not particularly programmatic. In 2016 you employ 26 separate str replace operations relating to the US on the same column. And you’re performing that operation on over 100000 rows! Instead of doing so many separate operations, building a pattern to capture many of US alternatives and then applying one str\_replace/str\_detect operation would be more computationally efficient. Using a case\_when/recode and setting the default to “all other countries” would be less repetitive than replacing all of the “other country” patterns, one by one. There are times when you don’t escape “.” characters - this can be dangerous because in regex . has a special meaning: match any character, remember to translate this into the ‘literal’ character with \\. - this will now match the . character only.

Similar in 2017 - the country cleaning is extensive but not programmatic. A lot is also repeated from the 2016 cleaning. If you find yourself having to copy and paste sections of code a lot, perhaps it would be worthwhile to write a function. The function could then be applied to any future data - what would you do if the business came to you with data for 2018 (or more) and asked you to incorporate it into your analysis. At present you’d have to write a bespoke cleaning script for it, rather than applying a few functions.

Your final bind\_rows operation could be changed to:

bind\_rows(list(clean\_2015, clean\_2016, clean\_2017))

as you don’t actually do anything more with clean\_15\_16.

Variable names: candy\_2015 is more informative than data\_2015, it’s good practice to call the data what the data is about, not data.

Analysis

Your analysis file reads like a file in development. I’d expect to see some text introducing the data and the results of the summary EDA (exploratory data analysis). I’d expect some description similar to.... The clean data consisted of 818680 of 7 variables `year\_id`, `trick\_or\_treating` ...

At present you use head() and tail() but don’t say: why, what data you're performing this on, or describe the output.

It’s fine to have bullet points of operations but maybe introduce them:

Assumptions

Assumptions made during data cleaning operations:

* Have only used the the data showing age, gender, if they going trick or treating, the type of sweets, their rating for the sweets and what country they have put down.
* Took out non sweets from the data as not sweets: 2 DVD box sets, vals of corn syrup, white bread, whole wheat, pencils, a board game, hugs form the person. This change the total number of vote for the 3 year by 32,036. From 727,573 individual votes to 695,037 votes.
* ...

Cleaning Process

1. use a script for the cleaning
2. cleaned the columns name
3. ...

Write filter(!is.na(rating)) rather than filter(rating != is.na(rating))

General practice: you often group by and then perform a filter operation. This is confusing to read as it makes the reader think that the group\_by is changing the data before the filter, which isn’t the case. This is more confusing for Q1, where you group, filter summarise, ungroup, and then summarise everything.

candy %>%

filter(!is.na(rating)) %>%

summarise(total\_rating\_num = n())

Would have achieved the same result.

Again for Questions 2 and 3:

candy %>%

filter(trick\_or\_treating == "Yes") %>%

summarise(avg\_age = mean(age)) %>%

floor()

Would arrive at the same result, without the additional group by.

I liked your solutions and thoughts to describing the most popular candies, as most popular could be interpreted in multiple ways. It’s important to say this though, and talk about the process you went through to find your definition of most popular….

To determine the most popular candy first the total number of positive ratings was calculated, the total number of negative ratings for each candy was then subtracted.

You get a lot of warnings running your analysis file:

## Warning in country == c("all other countries", "Unknown Country"): longer object

## length is not a multiple of shorter object length

What this warning is telling you: R is trying to do vector recycling:

(1,2,3,4,5) == (2,3)

R will compare element-wise. But since one vector is shorter than the other it will do something interesting when it runs out of elements in the shorter vector…. I.e. it will do 1 == 2, 2 == 3. At this point it has run out of elements in the shorter vector. R doesn’t stop trying to perform the comparison. It instead starts to cycle through the shorter vector. 3 == 2, 4 == 3, 5 == 2.

This will return FALSE. This isn’t the behaviour we’d like though. To see if elements from one vector are present in another vector use: %in%

(1,2,3,4,5) %in% (2,3)

Will return TRUE, which matches intuition.

The warning occurs when the length of the shorter vector can’t divide evenly the length of the longer vector. In my example we’d see this warning as well as 5/2 has a remainder of 1. This is important because in some cases we will have vectors of a length that will divide evenly the longer vector length, which wouldn’t show a warning even though vector recycling was still taking place.

Tl;dr - use %in% to test if elements of a vector are present in another vector.