**Data Wrangling Steps for Capstone Project**

This is a summary report of the most important steps I took to clean up the “Trending YouTube Video Statistics” dataset.

***Changing the category\_id variable to be more readable***

The original dataset already contained a “category\_id” variable that assigned each video a number, representing a particular category such as 1 = “Film & Animation”. Using RStudio, I used the **mutate** function in dplyr package to make the data more readable by adding a column (that I called “categories”) with the video category label for each video included in the dataset.

***Changing the trending\_date variable to separate out year, day, and month***

The original dataset contained a “trending\_date” variable that had the year, day, and month in this format: 18.25.05, which would stand for May, 25, 2018. Using RStudio, I used the **separate** function in dplyr package to break this variable up into three columns (one for year, one for day, and one for month).

The new columns were labeled “trending\_year” instead of “year” because I would need to use “year” again for the “publish\_time” variable. Because the “trending\_year” is only two digits, I had to change it to four digits to match the “publish\_year” column using the **mutate** function.

***Changing the publish\_time variable to separate out year, month, and day***

The original dataset contained a “publish\_time” variable that had the year, month, and day along with a timestamp in this format: 2018-04-20 10:40:51, which would stand for April, 20 2018 at 10:40 a.m. and 51 seconds. Using RStudio, I used the **anytime** function in anytime package to create a new column called “publish\_date” that allows me to then use the **separate** function in dplyr package to break this variable up into three columns (one for year, one for month, and one for day). This gives me both trending and publish variables with year, day, and month in their own columns. The timestamp was removed from the original “publish\_time” variable because the “trending\_date” variable did not have the time, so I don’t need it to compare the two variables going forward. Note that the order for the ‘day’ and ‘month’ are reserved for the trending and publish variables.

To then compare the two time variables, I created a “trend\_date\_official” and “publish\_date\_official” variable using the **with** and **anytime** functions.

***Changing the tags variable***

To code the “tags” variable, I first made a new data frame called “USvideos\_top100\_most\_viewed” of the top 100 viewed videos. Instead of finding the tags of all 40,949 observations, I thought it made more sense to only look at the tags used for the most popular videos, in this case defined as those videos that received the highest number of views. I used the **select** function to pull out the variables I wanted to keep from the cleaned data frame, which included “title”, “tags”, “views”, “likes”, “comment\_count”, “category\_id”, “categories”, “publish\_date\_official”, and “trend\_date\_official”. I then used the **distinct** and **mutate** functions to create a new column variable called “Tags”. I also used the **arrange** function to list the videos in descending order according to “views” – the top total view count.

Because the top 100 viewed videos represent only a couple of categories, this provides limited value, particularly to those clients whose videos do not fall into these more popular categories. Thus, it’s best to structure the data by “category\_id” and then find the top 100 viewed videos. To filter by “category\_id”, I used the **filter** function in which I set the “category\_id” variable = to a number that represented a specific video category. I did this step for each of the 32 categories.

To separate out each individual tag into its own column and keep it in its corresponding row to match the video the tag belonged to, I created a new data frame called “Tags\_Separated” from the “USvideos\_top100\_most\_viewed” data frame. Then I used the **separate** function to create columns for each tag belonging to each video. To remove the quotes from around the tags, I used the **gsub** function.

***To find the frequency for each tag***

In order to know which tags to recommend a potential client to use who wants to know which tags drive the most views, I needed to find out which tags were repeated most often. It should be noted here that I noticed some tags are similar but different. For example, one tag is ‘ed sheeran’ and another tag is ‘edsheeran’. This will result in two different tags each counted once instead of one tag being counted twice. In fact, some tags are misspelled. It’s too hard to tell if this was done intentionally to match the common search terms viewers use to look up a video. A judgment call will need to be made to decide how to handle this issue. On the one hand, it makes sense to use the tags as they were originally entered; however, the frequencies may be quite low, and thus not very useful, if they are not combined with similar tags.

To count the frequencies with which each tag appears, I created a value for each tag using the **table** and **unlist** functions. Then I wanted to summarize all of the tag frequencies into a single table. This was done using regular expressions.