Assignment 3: Emoji Zipfian

In this assignment you will use regular expressions to count Emoji occurences and plot their distribution.

1 New Emoji list

I have scraped a new Emoji list from a different source that is somewhat more suitable for our purposes. Please download **here** and store it in your project folder.

2 Count Emojis

2.1 Identify Emojis

Load `data_stream` and (new) `emoji_ids`. Iterate over Emojis and test whether Emoji is in text. Use, e.g., `grepl()` on collapsed text of tweets (`paste(text, collapse = ' ')`).

Hint: Use the column containing the Emojis not the Unicode string.

2.2 Count Emojis

Iterate over identified Emojis and count how often each of them occurs. Use, e.g., `gregexpr()`.

3 Expand analysis

New streams

Now that you knwo the drill. Rerun your code of Assignment 1 for *three* different track terms for *five* minutes (aka 300s) each.

Store data streams in your project folder.

Count Emojis

Then rerun analysis of section 2 (Count Emojis) for each of the three new data streams.

Store counts in your project folder.

4 Plot

Plot the three sets of Emoji counts (y-axis) for the three data streams against their Unicode strings (x-axis) from largest to smallest count.

Choose your adventure

Novice Plot one set of counts using high level `plot()` function. Decide whether to plot points or lines using the `type` argument. Control axis labels using `xlab` and `ylab`. Control size of points, labels, and axes through various `cex` arguments. Make sure `xlim` and `ylim` fit the other sets of counts. Add other sets of counts using `points()` or `lines()`. Distinguish the three sets using different `pch` or `lty`, respectively. Add legend using `legend()`.

Expert:base Setup a canvas using `plot.new()` and `plot.window`. Plot sets of counts using different `points()` or `lines()`. Add labels and axes using `mtext()`. Add legend using `legend()` or by hand using, e.g., `text()`, `lines`, `points`, and `rect`.

Expert:ggplot Figure it out. Very powerful, but not my cup of tea.

Add Zipf's Law to plot

Add additional line or points representing Zipf's law. Specifically, plot

$$f(rank) = \frac{1}{(rank + \beta)^{\alpha}}$$

using $\alpha \approx 1$ and $\beta \approx 2.7$.

To do this define function that returns $f(rank)$ as a function of $rank$, α, and $beta$. Then create a sequence from 1 to the number of Emojis in the plot and compute $f(rank)$. Add result to the plot.

Store plot using `png()`

Use `png()` to store your plot on the harddrive. To do this execute `png()` before executing your plot code. Feed it with a `file`name (with proper extension, i.e., .png) and dimensions (`height` and `width`).

After executing the device function (`png()`), and your plot code, execute `dev.off()` to finalize the plot.

Publish plot

Post your final plot on *twitter* using *#nlpbasel*.

Bonus level

Plot Emojis as x-axis labels. Download Emoji images **here**. Use function below to add Emoji png images.

Install and load `png` package. Identify Unicode strings of the to be plotted Emojis. Use `list.files()` to retrieve all Emoji filenames (Note: `full.names = TRUE` returns full path). Select filenames of to be plotted Emojis using regular expressions.

Hint: Emoji's can only be plotted into `plot region` (see `mai` in `?par`). Hint: If you used ggplot, check out **emoGG**.

```
# filename - character, complete file path to emoji png
# x - numeric, x position in the plot
# y - numeric, y position in the plot
# cex - numeric, size in plot units
# match - logical, should Emoji image match canvas aspect ratio.
add_emoji = function(filename, x, y, cex, match = F){
  pic  = readPNG(filename)
  dims = dim(pic)[1:2]
  usr = par()$usr
  if(adjust == T) ar = diff(usr[3:4]) / diff(usr[1:2]) else ar = 1
  rasterImage(pic, x-cex/2, y-(ar*cex/2), x+cex/2, y+(ar*cex/2), interpolate=TRUE)
  }
```