Documentation: plotte.R

Table of Contents

The Script

The R script documented here doesn't have such a streamlined structure as the **Collecto.R** script. Each section

Packages

As of writing this script and documentation, we only use two packages: * ggplot2 (Version 2.2.1) * gridExtra (Version 2.3)

The ggplot2 package

ggplot2 is a powerhouse of a package, when it comes to data visualisation. Our usage is rather basic and limited, however it certainly is able to create much more elegant graphics than R's default `plot()` command, which we will also use in this script at some point. From the ggplot2 package, we combine following functions:

```
ggplot()              # initializing the actual plot
aes()                 # greate "aesthetic" mappings in the plot object
geom_histogram()      # declaring the histogram style of the plot
scale_x_datetime()    # positioning scales for date and time
scale_y_continuous()  # positioning scales for continous data / index
ggtitle()             # setting a title for the plot
scale_fill_gradient() # give the visualized data a gradient color
```

The gridExtra package

gridExtra will only be used to arrange several plots produced by the `ggplot2` package next to each other, as this does not work with the `par()` function, commonly used in conjunction with R's default `plot()`. So, we only need gridExtra for ggplot2-objects:

```
grid.arrange() # arrange two ggplot2 objects in a grid
```

Participation by Platform

The *by-platform-graphic* actually consists of two plots, arranged next to each other. One side simply divides the collected data between the two categories "Twitter" and "Fediverse". This is especially easy to divide,

since the data we collected already comes in two discrete datasets for both platforms. Knowning this, we can simply create a factor variable `platform`, which contains the string `twitter` exactly so many times as we have tweet. The same is true for `fediverse`. for this, we use the `rep()` (repeat) as well as the `factor()` functions. The appropriate code looks like this:

```
twitter_number <- rep(x = "twitter", times = length(twitter$text))
fediver_number <- rep(x = "fediverse", times = length(mastodon$text))
platform <- factor(c(twitter_number, fediver_number),
                   levels = c("fediverse", "twitter"))
```

This data can now be visualized in a barplot later on.

The second plot seperates all fediverse-data into the single instances. Our scraped data contains the account name of each poster, which usually includes the instance-domain as well, for example: `fsfe@status.fsfe.org`.

In order to only extract the domains of the instances, we use the `sub()` function in conjunction with regex and save the results into the `instances` variable:

```
instances <- sub(x = as.character(mastodon$acct), pattern = ".*\\@", replace = "")
```

However, all accounts on the instance you scraped your data from - in this case from mastodon.social - only the username is displayed, not the domain of the instance. For example: `fsfe`.

In order to catch these as well, we look for all strings, that do not contain an `@` symbol with the `grep()` function and save their position into a variable (here: `msoc`). The `invert = TRUE` argument makes sure, that we get exactly those accounts, that do **not** contain the searched pattern:

```
msoc <- grep(x = as.character(mastodon$acct), pattern = "@", invert = TRUE)
```

Now we can replace all positions in the `instance` variable with the domain of the instance we scraped our data from. Afterwards, we should change the mode of the `instances` variable to `factor()`:

```
instances[msoc] <- "mastodon.social"
instances <- as.factor(instances)
```

Finally, we can start plotting. For this we use the default `plot()` function, as well as `legend()` to provide some extra information, necessary to understand the graphic. Our first plot uses the previously constructed `platform` variable as input. Since this is a factor variable, R will automatically create a barplot from this data. For the color, we use red for the Fediverse and blue for Twitter. The y-axes limit is a construction which *should* work in the future as well, but may need some minor adaption eventually. In its current form, it looks for the highest occurrence of tweets on a platform and rounds it up to the next higher 100 (110 would become 200, 401 would become 500).

```
ylim = c(0, ceiling(max(table(platform))/100) * 100)
```

The color in the second plot is generated with the `rainbow()` function, which will simply output a full color-spectrum starting from and ending with *red* again:

```
col = rainbow(n = length(unique(instances)))
```

In order to have both plots in a single graphic next to each other, `par()` can be used prior to plotting. The argument in use here generates a grid with one line and two columns:

```
par(mfrow=c(1,2))
```

By issuing `pdf()` prior to plotting and `dev.off()` afterwards, we can export the graphic directly to a PDF file (vectorized).

Participation by Time

For our second graphic we use functions from the `ggplot2` package instead of R's default `plot()` function, simply because `ggplot2` is much better with timeseries data.

Before doing so, we first have to create the timeseries for which we use the `date` and `time` variables in our datasets and the `strptime()`. Connecting these strings with `paste0()` (`paste()` would create a space bewteen both strings), they have the form: `YYYYMMDDhhmmss` which we also specify as the `format` argument:

```
twitter_time <- strptime(paste0(twitter$date, twitter$time),
                         format = "%Y%m%d%H%M%S")
mastodon_time <- strptime(paste0(mastodon$date, mastodon$time),
                          format = "%Y%m%d%H%M%S")
```

`ggplot2` has a rather unconventional syntax for R functions. You can combine several ggplot-functions with a `+`, enabling you to create extremely complex plots rather easily. The `ggplot()` function itself only initializes the plot object and specifies the data we are going to use. We combine (`+`) this function with `geom_histogram()` which - as the name might imply - creates the histogram itself. `scale_x_datetime()` and `scale_y_continuous()` specify the x-axes as timeline and y-axes as an index (counting continously). Lastly, we can specify a title of the plot with `ggtitle()` and fill the bars of the plot with a gradient color from low to high with `scale__fill_gradient()`. In the case of twitter, we save the entire plot into the `twitter_plot` variable for later use and do so similarly with mastodon/the fediverse:

```
twitter_plot <- ggplot(data = twitter, aes(x=twitter_time)) +
  geom_histogram(aes(fill=..count..), binwidth=60*180) +
  scale_x_datetime("Date") +
  scale_y_continuous("Frequency") +
  ggtitle("Participation on Twitter") +
  scale_fill_gradient("Count", low="#002864", high="#329cc3")
```

As opposed to R's `plot()` function, we can not use `par()` to create a unified graphic with `ggplot2`. Instead we use `grid.arrange()` from the `gridExtra` package. It takes both plots as arguments, as well as the number of columns it should arrange them in. With `pdf()` and `dev.off()` we can save the graphic into a PDF file (vectorized) directly:

```
pdf(file="./plots/ilfs-participation-by-date.pdf", width=14, height=7)
grid.arrange(twitter_plot, mastodon_plot, ncol = 2)
dev.off()
```