tubecleanR

Processing YouTube comment data in R

Julian Kohne Johannes Breuer

2025-08-15

At a glance

This tutorial introduces the `R` package `tubecleanR`, which provides a set of functions for cleaning and preprocessing *YouTube* comment data collected using the `tuber` or `vosonSML` `R` packages.

1. Introduction

YouTube is the largest and most popular video platform worldwide and its use generates vast amounts of data that can be of interest to researchers in the social sciences (and other fields). There are different types of *YouTube* data and these can be used to answer a large variety of research questions (see Breuer, Kohne, and Mohseni 2023; Deubel et al. 2024). One type of data that is of particular interest to researchers is the comments that users leave on videos.

YouTube comment data (as well as other types of data, such as video or channel statistics) can be collected in different ways (Deubel et al. 2024). An accessible, versatile, and widely used option is the YouTube Data API. The guide by Kohne et al. (2024) provides an introduction on "How to Collect Data with the YouTube Data API".

As is usually the case for text data (especially if they are collected from online platforms), *YouTube* comments are typically not directly "ready for analysis", particularly if quantitative text analysis methods are to be applied. Instead, they need to be cleaned and preprocessed first. The *KODAQS Toolbox* entry TextPrep by Peters and Shah (2024) provides a general introduction to text preprocessing in `R`. The `tubecleanR` package provides functionalities specifically for *YouTube* comment data.

There are various tools (`R` packages. `Python` libraries, or standalone software) that can be used for collecting *YouTube* comments (for an overview, see Deubel, Breuer, and Weller 2023; Deubel et al. 2024). Two popular options for collecting *YouTube* data from the "`R` world" are the packages `tuber` and `vosonSML`. These are the ones that the `tubecleanR` package focuses on.

2. Setup

Before you can use the package, you need to install it. The `tubecleanR` package is only available from *GitHub*, so you need to use the `remotes` package to install it.

```
remotes::install_github("gesiscss/tubecleanR", upgrade = "never")
```

After installing the package, you need to load it. For exploring the data, we will use the `dplyr` package from the `tidyverse` collection of packages.

```r
# Just for this notebook
library(DT)
library(skimr)

# For YT comment processing
library(tubecleanR)
library(tidyverse)

# For additional visualizations
library(wordcloud2)
library(dplyr)
library(tidyr)
library(tibble)
library(visNetwork)
```

If you want to use the `tubecleanR` package with your data, you first need to collect it using either `tuber` or `vosonSML`. Please note that both of these require API keys and `tuber` also requires an OAuth token. How to obtain these is explained in the guide by Kohne et al. (2024) as well as the publicly available materials for the 2023 GESIS Training workshop "Automatic Sampling and Analysis of YouTube Comments".

3. Application

Since the Terms of Service of the *YouTube Data API* do not allow to share the collected data, we cannot work with comment data from actual videos in this tutorial. Instead, we have simulated data using *Google Gemini*. The simulated data is contained in the `data` folder in the *GitHub* repository for this tutorial. This data follows the structure of the data returned by the `Collect()` function from the `vosonSML` package. Data collected with `vosonSML` includes further columns/variables. However, in the simulated data, we have only included the columns that are relevant for the `tubecleanR` package.

```r
simulated_yt <- read_csv2("data/simulated_yt.csv",
                          locale = locale(encoding = "UTF-8"))
```

Let's have a look at the structure of the simulated data.

```r
skim(simulated_yt)[,1:4] %>%
  datatable(options = list(pageLength = 25))
```

If we look through the strings contained in the `Comment` column, we will see that many of them contain emoji and some also contain URLs, user mentions, or video time stamps.

```r
simulated_yt %>%
  sample_n(20) %>%
  pull(Comment) %>%
  as.data.frame() %>% # just optics
  datatable() # just optics
```

> **Note**
>
> Please note that – just like the comments – the video IDs, URLs and user names in the data are simulated.

Now, let's use the `tubecleanR` package to clean and preprocess the data. The main function of the package is `parse_yt_comments()` which takes a dataframe containing *YouTube* comments collected with `tuber` or `vosonSML` as input and outputs a processed dataframe in which URLs, video timestamps, user mentions, emoticons, and emoji have been extracted from the comments into separate columns. In addition to that, the function creates a further column containing textual descriptions of the emoji, and another one containing a "clean" version of the comment in which the elements listed before as well as numbers and punctuation have been removed.

Of course, you can or - depending on your research question and analysis methods - might have to, run further preprocessing steps on the data, such as lowercasing, removing stop words, stemming or lemmatization, or tokenization. The `tubecleanR` package does not provide functions for these steps, but you can, e.g., use the `tidytext` package for that (the *KODAQS Toolbox* entry TextPrep also provides some further guidance on these steps).

> **Note**
>
> Please note: As the preprocessing involves multiple steps, running the `parse_yt_comments()` function can take a while, especially if you have a large dataset. The simulated data we are using here is small, so it should not take long to process.

```
# here, we also specify that the data was collected with the vosonSML package
processed_yt <- parse_yt_comments(simulated_yt,
                                  package = "vosonSML")
```

Let's have a look at the structure of the processed data.

```
skim(processed_yt)[,1:4] %>%
  datatable(options = list(pageLength = 25))
```

We can compare the `OriginalText` with the newly created `CleanedText` column to see how the comments have been cleaned.

```
processed_yt %>%
  select(OriginalText, CleanedText) %>%
  sample_n(20) %>%
  as.data.frame() %>% # just optics
  datatable() # just optics
```

In addition to extracting them, the `parse_yt_comments()` function also creates a column containing textual descriptions of the emoji included in the comments using a dictionary approach.

```
processed_yt %>%
  filter(!is.na(Emoji)) %>%
  select(Emoji, EmojiDescription) %>%
  sample_n(20) %>%
  as.data.frame() %>% # just optics
  datatable() # just optics
```

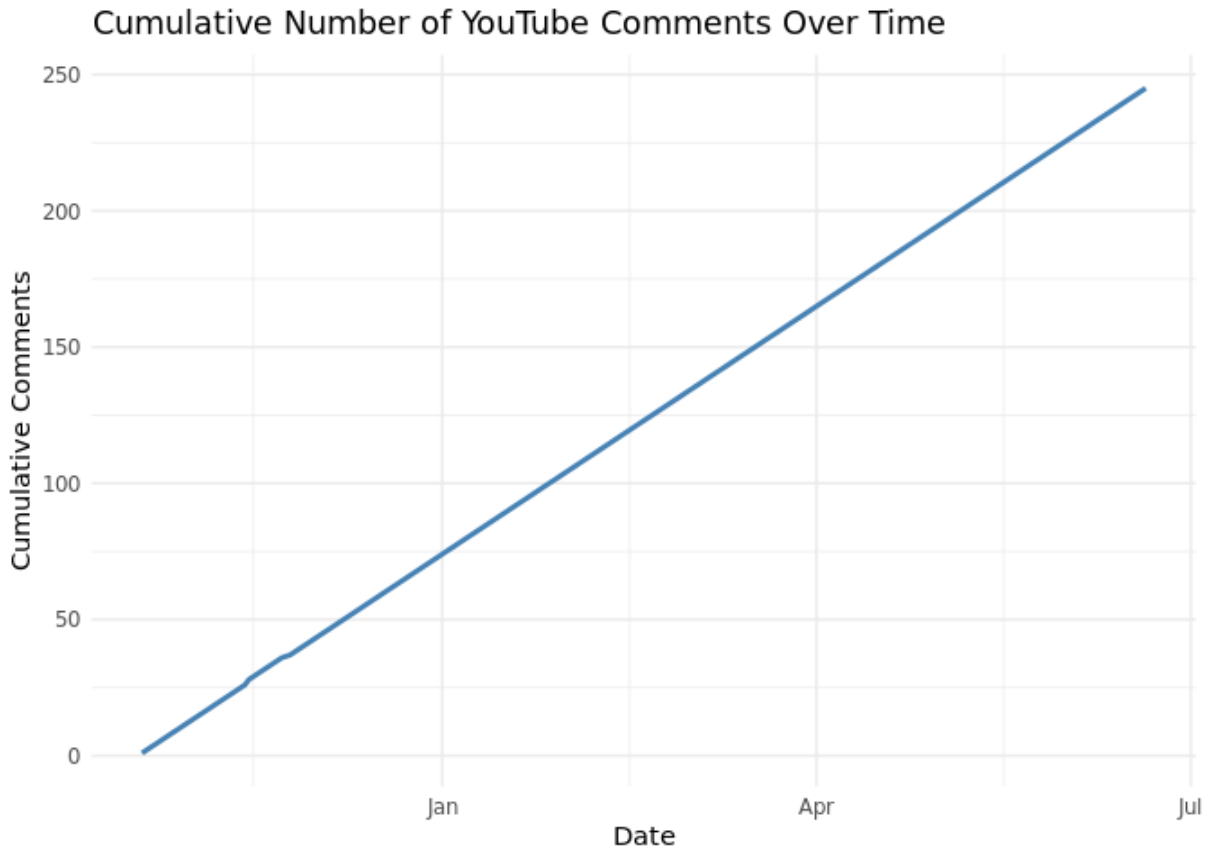
You can find the dictionary as an attached csv file in the `tubecleanR` package. Please keep in mind that the dictionary might get outdated as new emoji are added over time. However, you can simply append the csv file in the package to add new emoji to the dictionary based detection.

4. Visualizations and further Analysis

We can now build on the preprocessed comments to further explore and analyze the YouTube comment data. For example by:

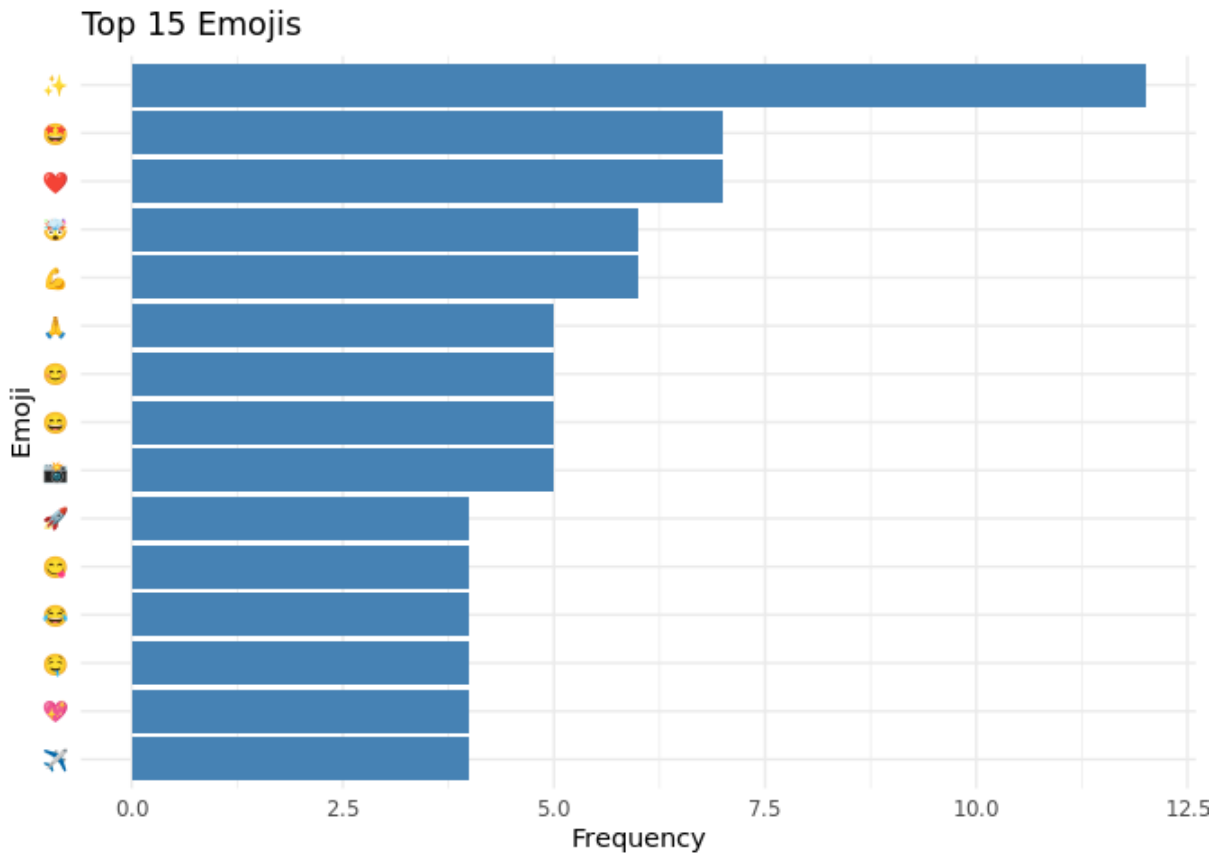
Plotting a timeline of comments

```
# Comment timeline
processed_yt |>
  filter(!is.na(Published)) |>
  mutate(date = as.Date(Published)) |>
  count(date) |>
  arrange(date) |>
  mutate(cumulative_comments = cumsum(n)) |>
  ggplot(aes(x = date, y = cumulative_comments)) +
  geom_line(color = "steelblue", size = 1) +
  labs(
    title = "Cumulative Number of YouTube Comments Over Time",
    x = "Date",
    y = "Cumulative Comments"
  ) +
  theme_minimal()
```



Plotting the distributions of Emoji

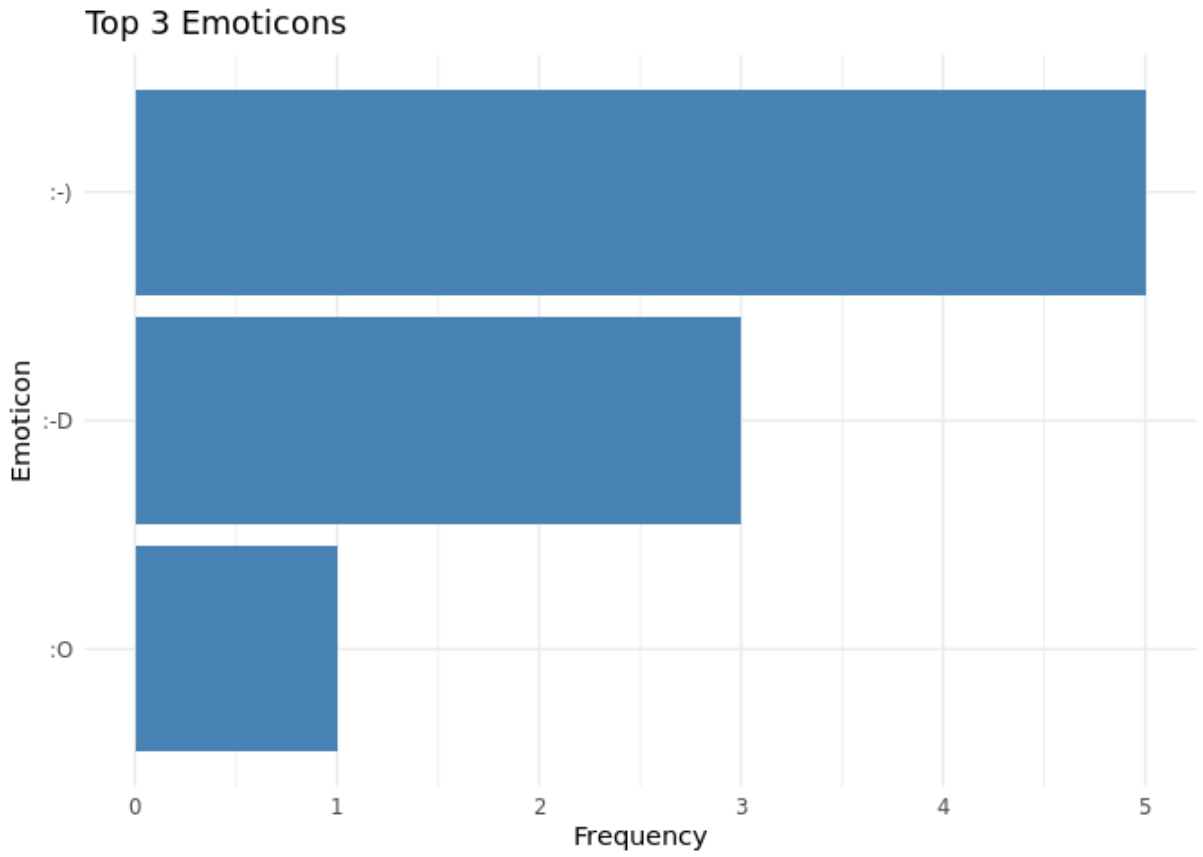
```
# Emoji Distributions
processed_yt |>
  unnest_longer(Emoji) |>
  filter(!is.na(Emoji)) |>
  count(Emoji, sort = TRUE) |>
  slice_max(n, n = 15) |>
  ggplot(aes(x = reorder(Emoji, n), y = n)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Top 15 Emojis", x = "Emoji", y = "Frequency") +
  theme_minimal()
```



Plotting the distribution of Emoticons

```
# Emoticon plot
processed_yt |>
  unnest_longer(Emoticons) |>
  filter(!is.na(Emoticons)) |>
  count(Emoticons, sort = TRUE) |>
  slice_max(n, n = 3) |>
  ggplot(aes(x = reorder(Emoticons, n), y = n)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
```

```
  labs(title = "Top 3 Emoticons", x = "Emoticon", y = "Frequency") +
  theme_minimal()
```



Plotting a Wordcloud

```
# Wordcloud
words <- tolower(processed_yt$CleanedText) |>
  paste(collapse = " ") |>
  strsplit("\\W+") |>
  unlist()
words <- words[nchar(words) > 2]
freq_tbl <- sort(table(words), decreasing = TRUE)
df <- data.frame(word = names(freq_tbl), freq = as.numeric(freq_tbl))

# Render interactive, dense HTML word cloud
set.seed(123)
wordcloud2(df, size = 2.5, color = "random-dark", backgroundColor = "white")
```

Plotting a network of user-mentions

```
# Network plot

# Create edges from Author to mentioned users
edges <- processed_yt |>
```

```r
  unnest_longer(UserMentions) |>
  filter(!is.na(UserMentions), Author != UserMentions) |>
  count(Author, UserMentions, name = "value") |>
  rename(from = Author, to = UserMentions)

# Create nodes from all unique users (authors + mentions)
all_users <- unique(c(edges$from, edges$to))
nodes <- tibble(id = all_users, label = all_users)

# Plot interactive network
visNetwork(nodes, edges) |>
  visEdges(arrows = "to") |>
  visOptions(highlightNearest = TRUE, nodesIdSelection = TRUE) |>
  visLayout(randomSeed = 42)
```

5. Discussion

YouTube user comments can be a very interesting data source for social science research (Breuer, Kohne, and Mohseni 2023; Deubel et al. 2024). There are various options for collecting such data (Deubel, Breuer, and Weller 2023; Deubel et al. 2024). Many of the available tools make use of the *YouTube Data API* (see Kohne et al. 2024 for an introduction). This also applies to the `tuber` and `vosonSML` packages for `R`.

The `tubecleanR` package provides a set of functions for cleaning and preprocessing *YouTube* comment data collected with these two packages. In addition to extracting URLs, video timestamps, and user mentions, it also extracts emoticons and emoji from the comments and creates textual descriptions of the emoji. Emoji have often been ignored in analysis of social media data, but they can be an important part of the communication and meaning in online comments (Breuer, Kohne, and Mohseni 2023). The `tubecleanR` package provides a simple way to extract and preprocess them.

The functionalities provided by the`tubecleanr` package facilitate the cleaning and preprocessing of *YouTube* comment data and can, thus, contribute to increasing the quality of such data. The prepossessed data can be used by researchers for quantitative and/or qualitative analysis, e.g., using text mining or natural language processing (NLP) methods.

6. Literature

Breuer, Johannes, Julian Kohne, and M. Rohangis Mohseni. 2023. "Using YouTube Data for Social Science Research." In *Research Handbook on Digital Sociology*, edited by Jan Skopek, 258–77. Edward Elgar Publishing. https://doi.org/10.4337/9781789906769.00022.

Deubel, Annika, Johannes Breuer, Julian Kohne, and M. Rohangis Mohseni. 2024. "Overview of Working with Data from YouTube (GESIS Guides to Digital Behavioral Data, 12)." GESIS - Leibniz-Institute for the Social Sciences. https://doi.org/10.60762/GGDBD24012.1.0.

Deubel, Annika, Johannes Breuer, and Katrin Weller. 2023. "Collecting Social Media Data: Tools for Obtaining Data from Social Media Platforms." 1. Center for Advanced Internet Studies. https://www.cais-research.de/wp-content/uploads/Collecting-Social-Media-Data.pdf.

Kohne, Julian, Johannes Breuer, Annika Deubel, and M. Rohangis Mohseni. 2024. "How to Collect Data with the YouTube Data API (GESIS Guides to Digital Behavioral Data, 13)." GESIS - Leibniz-Institute for the Social Sciences. https://doi.org/10.60762/GGDBD24013.1.0.

Peters, Yannik, and Kunjan Shah. 2024. "TextPrep - Comparing Tools and Workflows for Data Quality in Basic Text Preprocessing with R." GESIS – Leibniz Institute for the Social Sciences. https://kodaqs-toolbox.gesis.org/github.com/YannikPeters/DQ_Tool_TextPreprocessing/index/.