Hashtags on TikTok - data collection analysis and visualization

In this article we will go through the different steps necessary to conduct a hashtag analysis on TikTok that can result in a visualization like this:
https://martin.degeling.com/snv/fyp-hashtags-by-week/

The basis of the visualization is a continous data collection of videos to TikTok.

TikTok primer, if you missed that: TikTok is an app based social network where users share (mostly) short videos on various topics. Over the last few years TikTok has been the fastest growing social network. Besides the focus on moving images another point that distinguished TikTok from other social networks is that it's feed algorithm (the recommender system) is less focussed on the social graph (showing videos that your contacts have liked or viewed), but instead creates interest graphs that focus on showing content regardless of the author.
Critique.
HEating etc.

The purpose of a hashtags analysis on the TikTok generic FYP is to learn more about what the videos are initially shown to users that have not used the service before.

The process of the analysis is as follows:

Step 1: Scraping Data with puppeteer
Step 2: Analyzing and Visualizing with Jupyter Notebooks

Scraping

For this analysis we will leverage the webversion of TikTok accessible with any browser on TikTok.com.
While there are various "unoffical APIs" or libraries available that allow you to scrape content from TikTok, the purpose of this chapter is to built a scraper from scratch.

For starters we are using pupeteer to open the webversion of the FYP.

Tech Setup

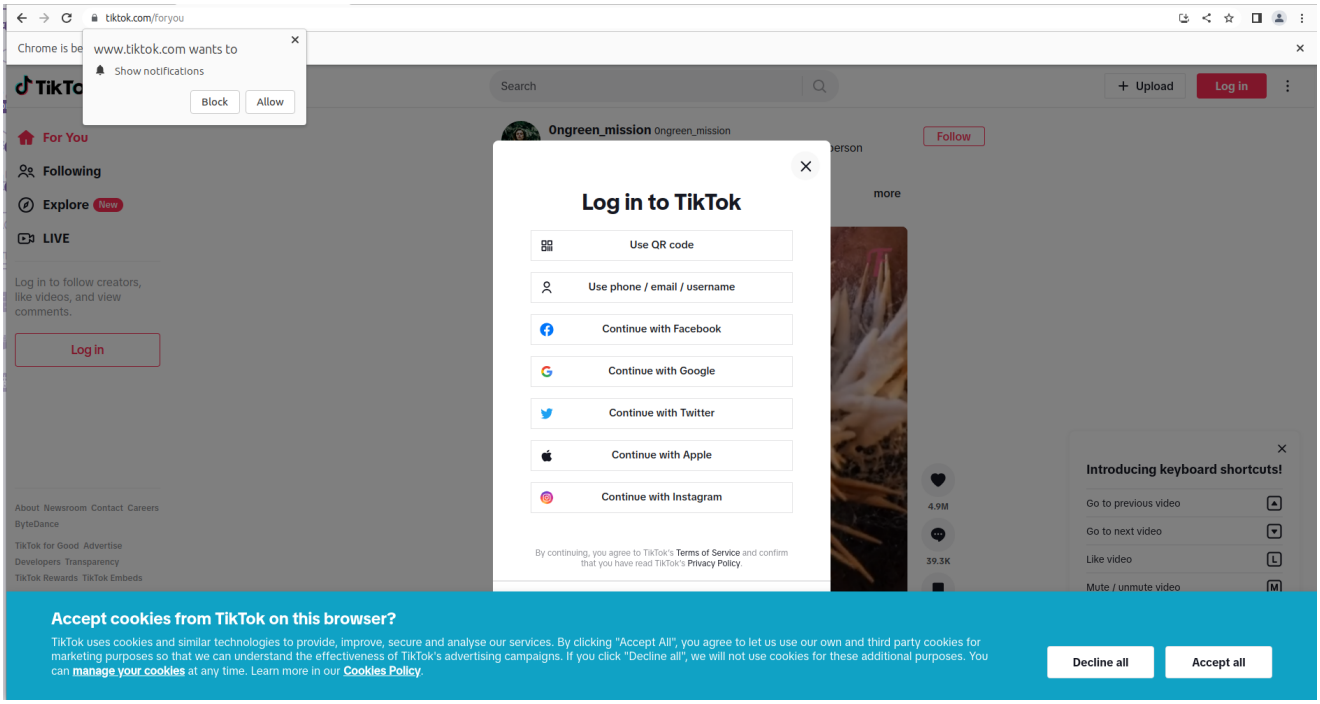
To run this project you should be able to open a command line on your computer and install software. For the first part you need to have nodejs installed. After you have created a folder and navigated to it with the console run `npm install pupeteer` in it.

A basic scraper that opens the TikTok website should look like this:

```
const puppeteer = require('puppeteer');
async function run() {
        const browser = await puppeteer.launch({
```

```
            executablePath: '/usr/bin/google-chrome',
            headless:false,
            defaultViewport: null,
            args: ["--window-size=1920,1080"]
        });
        const page = await browser.newPage();
        await page.goto('https://tiktok.com/foryou');
        }

run();
```

This should bring up a browser windows like this:

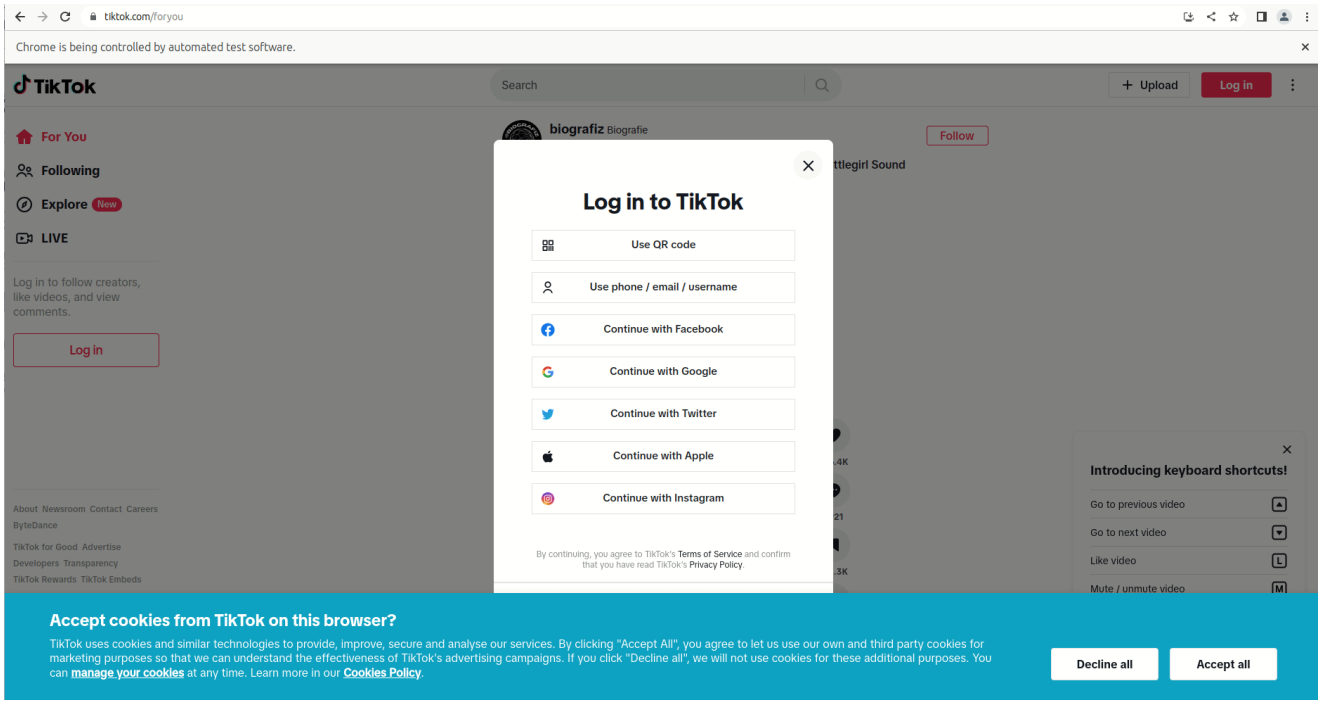


At this point we can already get some information about the videos tiktok would like to show us - even though we can't see them. Because it's already in the code.
You can right-click on the page an select "view source" (or press ctrl+u) to see the HTML source code of the site. Search for "ItemModule" and you'll jump to information about the videos in a structured json-format. The JSON object contains information about the first eight videos that are put on the FYP. The structure of each JSON object is shown here
We can extract that data automatically and put it on an internal list like this:

```
// once the page is loaded we will search for the javascript block with the ID
#SIGI_STATE and parse it's content as JSON
const sigistate = await page.$eval('#SIGI_STATE', el => el.innerHTML);
initinfo = JSON.parse(sigistate)
videos = []

// Iterate through the list of videos and add it to our own data structure
Object.keys(initinfo.ItemModule).forEach( async (el,c) => {
```

```
        videos.push(initinfo.ItemModule[el])
})
```

To get data on more videos we have to interact with the website. That doesn't work initiatilly because of the annoying login popup, but we can close it automatically



We can use the developer settings from chrome to find out how we can address the button with pupeteer. First open the developer console with F12 than right click the close icon, select "Inspect" and find the parentelement that seems unique. In this case there is a `div` element with the attribute `data-e2e="modal-close-inner-button"`
In pupeteer we can use this attribute to identify the X and click it like this

```
let logindialog = await page.$('div[data-e2e="modal-close-inner-button"]')
await logindialog.click(page, 'div[data-e2e="modal-close-inner-button"]');
```

After that we can scroll the page forever, tapping the key-down every two seconds

```
while (true) {
        await page.waitForTimeout(2000);
        await page.keyboard.press('ArrowDown');
}
```

When watching the bot scroll through TikTik and looking at the Network traffic in the developer console you'll notice that there is a infrequent call to a TikTok API https://www.tiktok.com/api/recommend/item_list/ that returns the list of the next 30 videos.
Finding this request and understanding what it is might be more complicated on other service. But undocumented APIs can be a key element of data driven investigations.
With pupeteer we can intercept all network Traffic - and since it is running from within the browser we do not have to worry about any encryption. We can intercept the responses to this specific URL and process the video metadata as we like:

```
await page.setRequestInterception(true);
page.on('request', (request) => request.continue())
page.on('response', response => {
try {
        if (response.url().indexOf("api/recommend/item_list/item_list") > 0){
                response.json().then((data) => {
                        data.itemList.forEach( async (el,c) => {
                                videos.push({id:el.id})
                        })
                })
                        console.log("Received Video Data. Seen " +
videos.length + " so far")
                }
        } catch(e) {
                console.log(e);
        }
});
```

You have not successfully created a scraper for the TikTok public ForYouPage with less than 50 lines of code
Get the full code *here*

Storing

To analyse the data set over a longer period of time we need to store and process it. For this example project we can use a file-based database like node-json-db that stores everything in a single json file. If you plan to run a project like this on a larger scale you should consider regular database servers like mongoDB oder Redis.
You can install it with `npm install node-json-db` similar to the installation of pupeteer above.
You can than run the full code from here

Analysis

Tech Setup

We are using jupyter notebooks with python. I recommend using miniconda to run python and install dependencies. You can use the conda enviroment published with this tutirial here.

Import Data

First we can import the JSON database into a python dictionary:

```
import json
with open('../../../tiktok-fyp-videos.json') as file:
    database = json.load(file)
print(len(database.keys()), "videos in database")
```

this will result in something like: `474 videos in database`

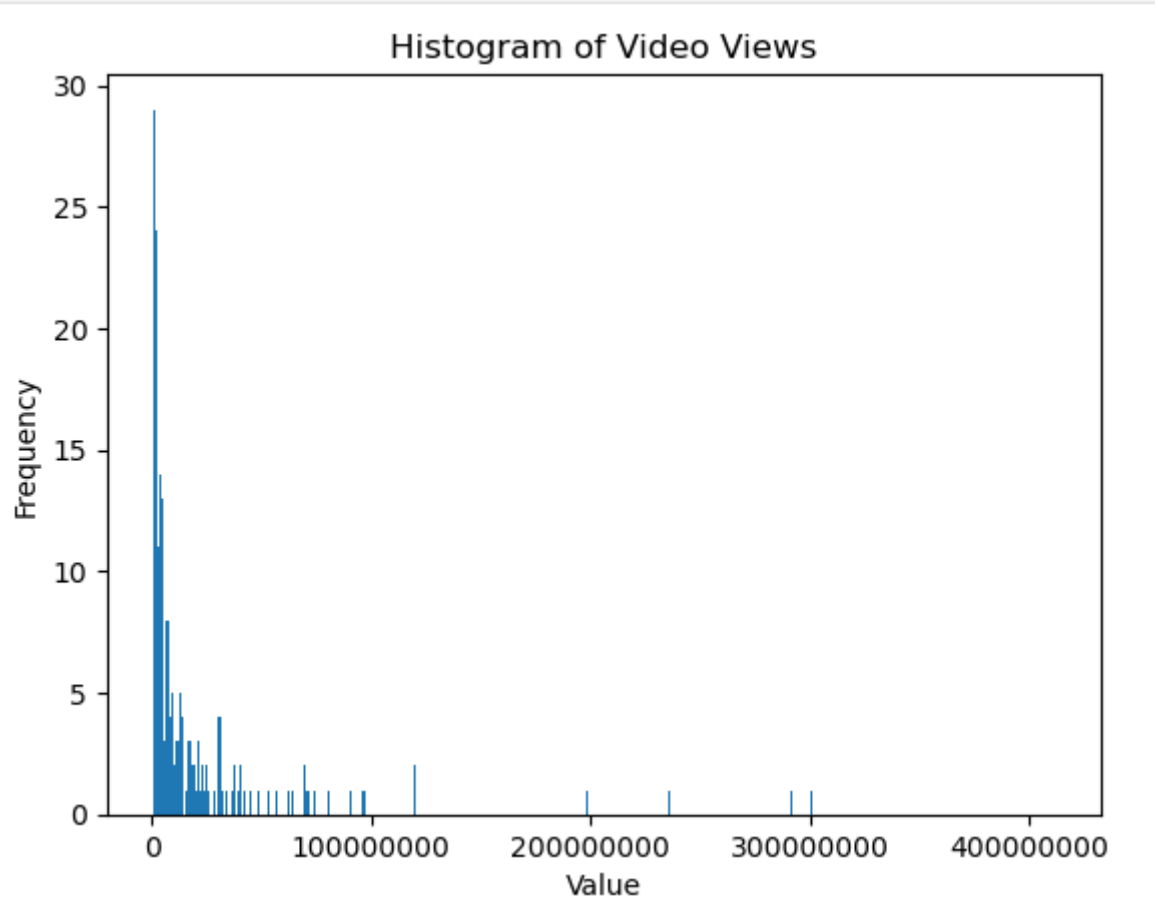
At this point we could have a look at the data the we actually captured.

```
import pprint
pprint.pprint(next(iter(database.values())))
```

Will print the first video in the database. It contains a lot of technical information, e.g. about the video quality and URLs of cover pictures, avatars, music and subtitles.
But also the name of the author in `authorID` some statics in `stats` and of course the hashtags in the `textExtra` field.
Let's first look at the video statistics. We first iterate over all videos and collect the views in an array. And then let the numpy package do the math:

```
video_views = []
for video in database.values():
    video_views.append(video["stats"]["playCount"])

average = np.mean(video_views)
print("Average views:", average)
```

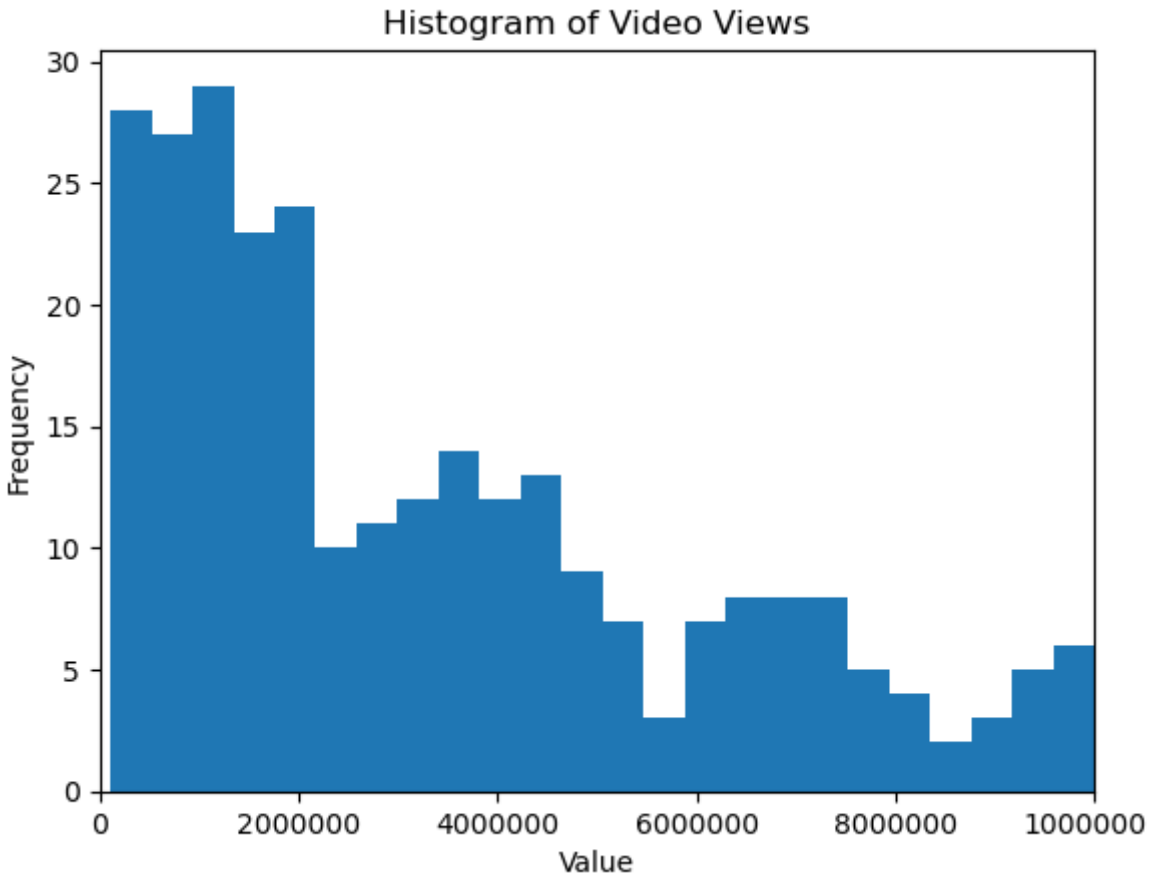
This will result in `Average views: 20519932.70042194`, more than 20 Million views per video on average? That seems like a lot. A histogram plot shows that the data is not evenly distributed. There are a few videos with a huge number of views. Outliers from the rest.



Because of this distribution the median will therefore give a better idea of the 'average'

```
median = np.median(video_views)
print("Median views:", median)
```

We can also limit the histogram plot to up to 10 Mio views and see that the majority of vidoes shown on the FYP for non-logged in web users is focused on videos that already had a large audience.



Hashtag Analysis

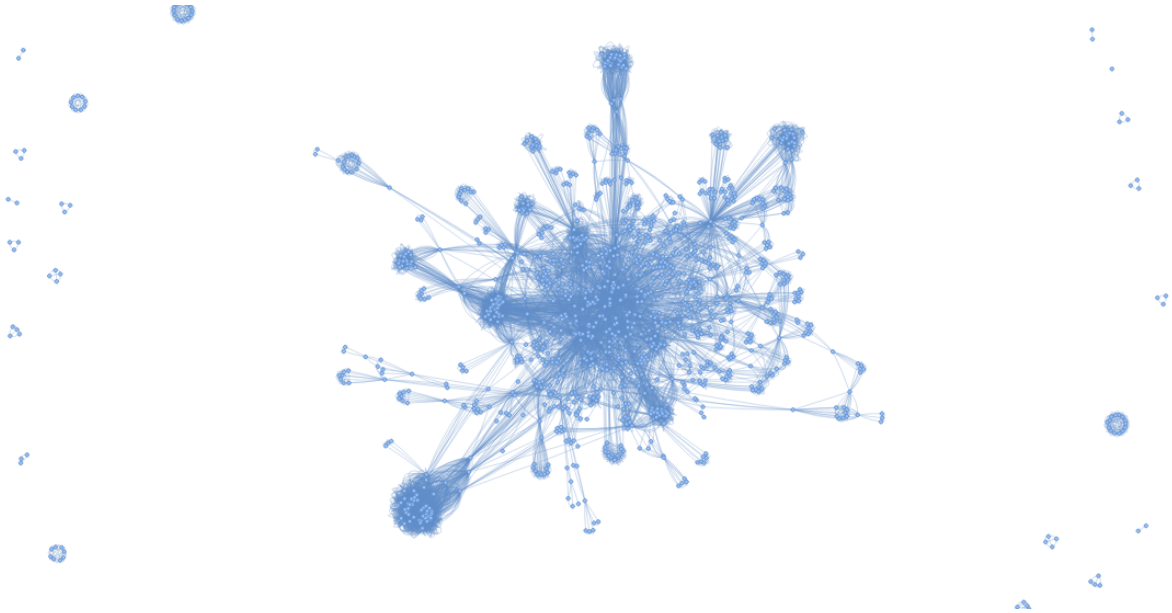
We can now iterate over all videos in the database to count hashtag occurrences and videos without hashtags to better understand the quality of the data.

```
from itertools import combinations
import numpy as np

videos_wo_hashtags = 0
hashtag_occurences = {}

for video in database.values():
    video_views.append(video["stats"]["playCount"])
    if "textExtra" in video:
        for hashtag in video["textExtra"]:
            if hashtag["hashtagName"] not in hashtag_occurences:
```

```
                hashtag_occurences[hashtag["hashtagName"]] = 1
            else:
                hashtag_occurences[hashtag["hashtagName"]] += 1
    else:
        videos_wo_hashtags += 1
# Use of hashtags
print("There are", videos_wo_hashtags, "videos without hashtags (",
videos_wo_hashtags*100/len(database.keys()), "%)")
```

We learn that 17% of the videos do not use hashtags.

If we go ahead and plot the network of hashtag combinations we will get a graph like this:



Similar to the views, we can also look at the distribution of hashtags, by sorting the list and printing the 20 most common hashtags.

```
hashtags_sorted = dict(sorted(hashtag_occurences.items(), key=lambda item:
item[1], reverse=True))
print("The ten most common hashtags:")
print(list(hashtags_sorted)[:20])
```

We see that the most common hashtags are related to tiktok itself and do not describe it's content. To learn more about what type of videos are shown on the FYP we should exlude these hashtags: ['fyp', '', 'viral', 'foryou', 'foryoupage', 'fy', 'fypシ', 'funny', 'funnyvideos', 'tiktok', 'fürdich', 'trending', 'trend', 'viralvideo']

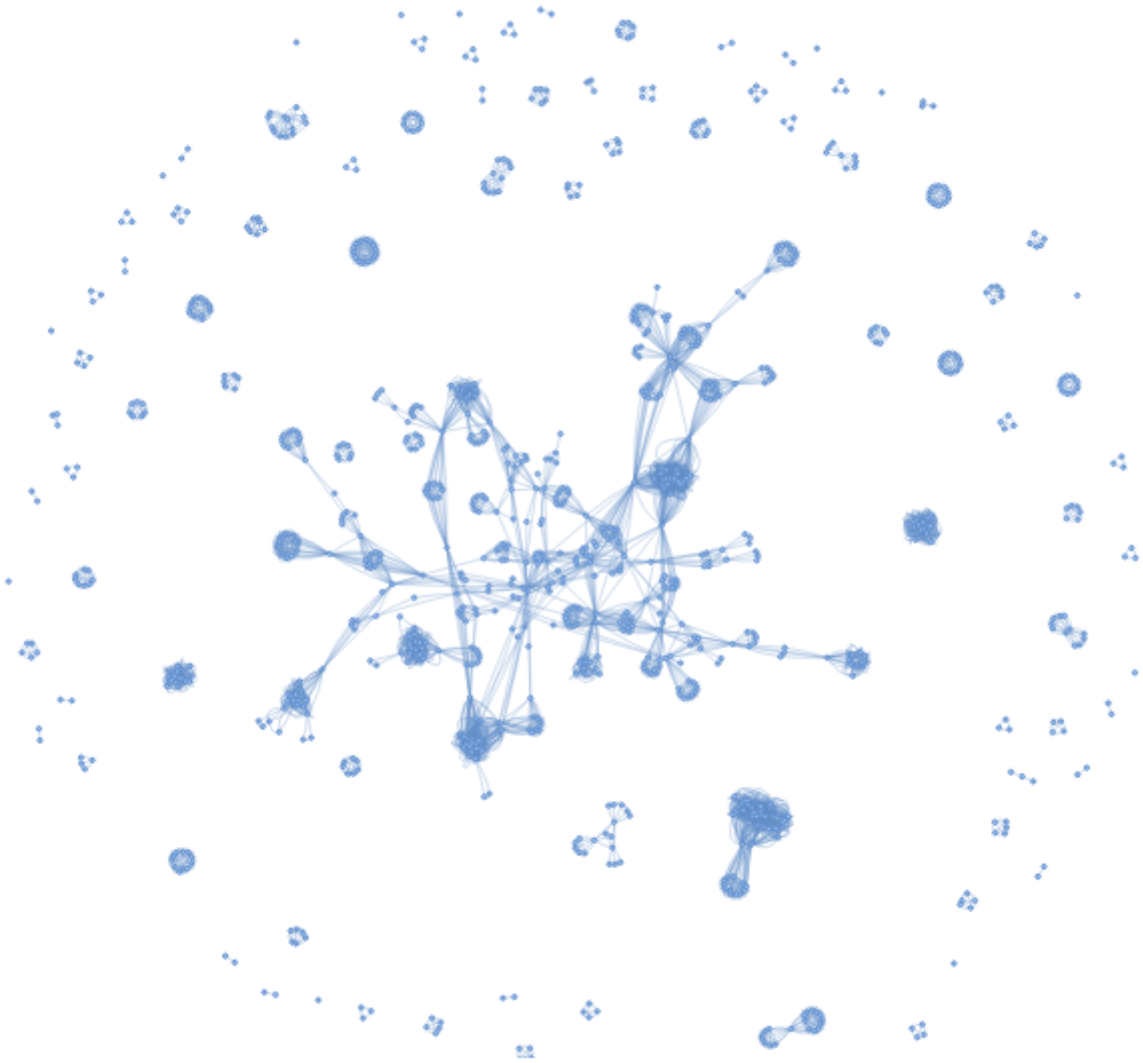
A network graph consists of nodes (the circles) and edges (connections). Again we iterate over all videos to get a list of hashtag (edges) and store as an edge if they occured together on a video.

```
exclude_hashtags = ['fyp', '', 'foryou', 'viral', 'foryoupage', 'fypシ', 'fy',
'fürdich', 'trending', 'foryoupage', 'tiktok', 'viralvideo', 'fürdichseiteシ',
'4u']
hashtag_nodes = {}
hashtag_edges = []
```

```
for video in database.values():
        listofhashtags = []
        if "textExtra" in video:
                for hashtag in video["textExtra"]:
                        if hashtag["hashtagName"] not in exclude_hashtags:
                                listofhashtags.append(hashtag["hashtagName"])
                                if hashtag["hashtagName"] not in hashtag_nodes:

hashtag_nodes[hashtag["hashtagName"]]=len(hashtag_nodes.keys())+1
        for combination in combinations(listofhashtags, 2):
                hashtag_edges.append([hashtag_nodes[combination[0]],
                hashtag_nodes[combination[1]]])
```

and visualize it with pyvis:

```
net = pyvis.network.Network(notebook=False, cdn_resources="local")

node_count = 1
for hashtag in hashtag_nodes:
        net.add_node(node_count, label=hashtag)
        node_count+=1
        net.add_edges(hashtag_edges)
        net.show_buttons(filter_="physics")
net.show('selected_hashtags.html')
```



TODO: Interpretation