Web Science Project Week 3+4 on Network Dynamics

Emiel Steegh - s1846388
Freek Nijweide - s1857746

Project Description

For this week's assignments, we will examine network dynamics, which concerns effects such as information cascades, network effects, information diffusion, API calls, and more.

We use several packages this week, please make sure these are installed when trying to run our code:

- pandas
- numpy
- plotly

We decided to use plotly instead of matplotlib (like in the last few weeks) due to its superior interactivity and clarity.

We will now begin by defining some global imports and functions.

For handling this week's data, we convert the JSON files to Pandas DataFrames. We store these in .csv files so that we do not have to rebuild them each time (which takes very long). This made the data much more manageable.

In [1]:

```python
# The following includes are needed to work with graphs and display solutions.
from __future__ import division
import networkx as nx

from IPython.display import SVG
from IPython.display import HTML
from IPython.display import display

import StringIO

from networkx.drawing.nx_pydot import read_dot
from networkx.drawing.nx_pydot import from_pydot
from networkx.drawing.nx_agraph import to_agraph
import pydot

import numpy as np
import random

import glob #used for filefinding
import pprint #debugging purposes
import pandas as pd #dataframes for plotly and .csv saving/reading
import os #path handling
import json

import dateutil.parser

try: #plotly does not come with the standard python and conda package
    import plotly.graph_objects as go
    import plotly.colors
except:
    print("\n"+
          "┌---------------- (!) Warning (!) ----------------┐")
    print("|                                                 |")
    print("| \x1b[31m  It looks like Plotly could not be imported,\x1b[0m   |")
    print("|   please make sure it is properly installed.    |")
    print("|                                                 |")
    print("└-------------------------------------------------┘")
else:
    print("\x1b[32mimports successful")
```

```
imports successful
```

In [2]:

```
#credits for the following function go to Benjamin Toueg
#https://stackoverflow.com/questions/16888409/suppress-unicode-prefix-on-strings-wh
en-using-pprint
def my_safe_repr(object, context, maxlevels, level):
    '''
    when using prettyprint prints unicode as normal strings instead of u'(str)'
    '''
    typ = pprint._type(object)
    if typ is unicode:
        object = str(object)
    return pprint._safe_repr(object, context, maxlevels, level)

pp = pprint.PrettyPrinter()
pp.format = my_safe_repr

def ANSI_col_code(color):
    '''
    changes a specified color or letter into it's ansii terminal color number count
erpart
    '''
    # for color codes refer to
    # https://en.wikipedia.org/wiki/ANSI_escape_code#Colors
    code = 31
    if type(color) not in [str, float]:
        code = 31
    elif color.lower() in ['r', "red"]:
        code = 31
    elif color.lower() in ['g', "green"]:
        code = 32
    elif color.lower() in ['b', "blue"]:
        code = 94
    elif color.lower() in ['y', "yellow"]:
        code = 33
    elif color.lower() in ['m', "magenta", 'p', "purple"]:
        code = 35
    return code

def cprint(string, color = 'r'):
    '''
    prints a string with a color through the use of ansii escape characters
    '''
    code = ANSI_col_code(color)
    print "\x1b[{}m{}\x1b[0m".format(code, string)

def cstring(string, color = 'r'):
    '''
    returns a string with a color through the use of ansii escape characters
    '''
    code = ANSI_col_code(color)
    return "\x1b[{}m{}\x1b[0m".format(code, string)

cprint("Red", 'r')
cprint("Green", 'g')
cprint("Blue", 'b')
cprint("Yellow", 'y')
cprint("Purple", 'm')
```

```
Red
Green
Blue
Yellow
Purple
```

Data read, parse & store

The code creates the necessary data from the many json files found in the supplied /data folders.
To save time the data is then stored in 4 seperate much smaller csv files which are read when the code is run. If they are not present, the .csv files will be created again.

It should be noted that the datasets that rely on youtube (youtube_top100, radio538_alarmschijf & radio3fm_megahit) miss information between the 4th of January until the 21st of March, this can be clearly seen in most graphs.

In [3]:
```
def filetype_paths (data_folder, filetype='json'):
    '''
    returns a list of paths of the *.json files in a dir (data_folder)
    '''
    target = os.path.join(data_folder, '*.{}'.format(filetype))
    paths = glob.glob(target)
    return paths

def path_to_timestamp(path):
    '''
    turns yyyymmdd into yyyy-mm-dd (the standard dateformat for plotly)
    '''
    dateraw = ((os.path.split(path)[1])[:8])
    date = "{}-{}-{}".format(dateraw[:4], dateraw[4:6], dateraw[6:8])
    return date


def data_from_youtube(folder):
    '''
    read a supplied list of json files
    the files contain a list of snapshots of data on a songs youtube page
    from each entry in each file the following information is taken and put in one
large dataframe
    time of data collection | youtube name | youtube ID | release date | views | li
kes+dislikes | likes | dislikes
    '''

    #grab the list of files to parse
    ls_locations = filetype_paths(folder, 'json')

    #grab the list of files to parse
    df = pd.DataFrame(columns=['timestamp', 'song', 'youtube_id',  'release_date',
'views', 'votes', 'likes', 'dislikes'])

    #dict conatining the different collumns for appending data
    new_row = {'timestamp' : "_", 'release_date' : "_", 'song' : "_", 'youtube_id'
: "_", 'views' : 0, 'votes' : 0, 'likes' : 0, 'dislikes' : 0}

    #loops through all files in the data folder
    for loc in ls_locations:

        #grabs datagather timestamp from file title
        new_row['timestamp'] = path_to_timestamp(loc)
        file_raw_data =json.load(open(loc))

        #take all the other data from the relevant locations in the file
        for entry in file_raw_data:
            new_row['song'] = (entry["snippet"])["title"]
            new_row['youtube_id'] = entry["id"]
            new_row['release_date'] = ((entry["snippet"])["publishedAt"])[:10]
            stats = entry["statistics"]
            new_row['views'] = int(stats["viewCount"])
            new_row['likes'] = int(stats["likeCount"])
            new_row['dislikes'] = int(stats["dislikeCount"])
            new_row['votes'] = new_row['likes']+new_row['dislikes']

            df = df.append(new_row, ignore_index=True)

    return df

def data_from_spotify(folder):
    '''
    read a supplied list of json files
    the files contain a list of the top songs on spotify that day with their respec
```

In [4]:
```
check_or_create_data()

testprint = False
if testprint:
    print "\n\n"
    cprint("--- --- Radio3fm Megahit:",'p')
    print data_megahit
    cprint("--- --- Radio538 Alarmschijf:",'p')
    print data_alarmschijf
    cprint("--- --- Spotify Top100 Dataset:",'p')
    print data_spotify100
    cprint("--- --- Youtube Top100 Dataset:",'p')
    print data_youtube100
```

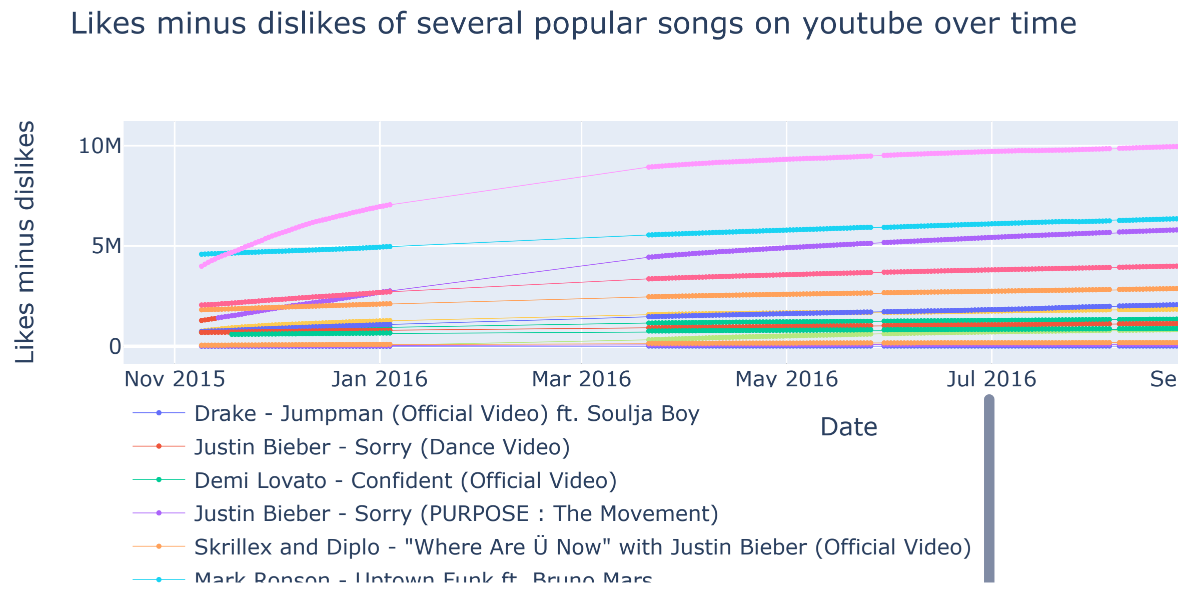
```
all data files seem present!
```

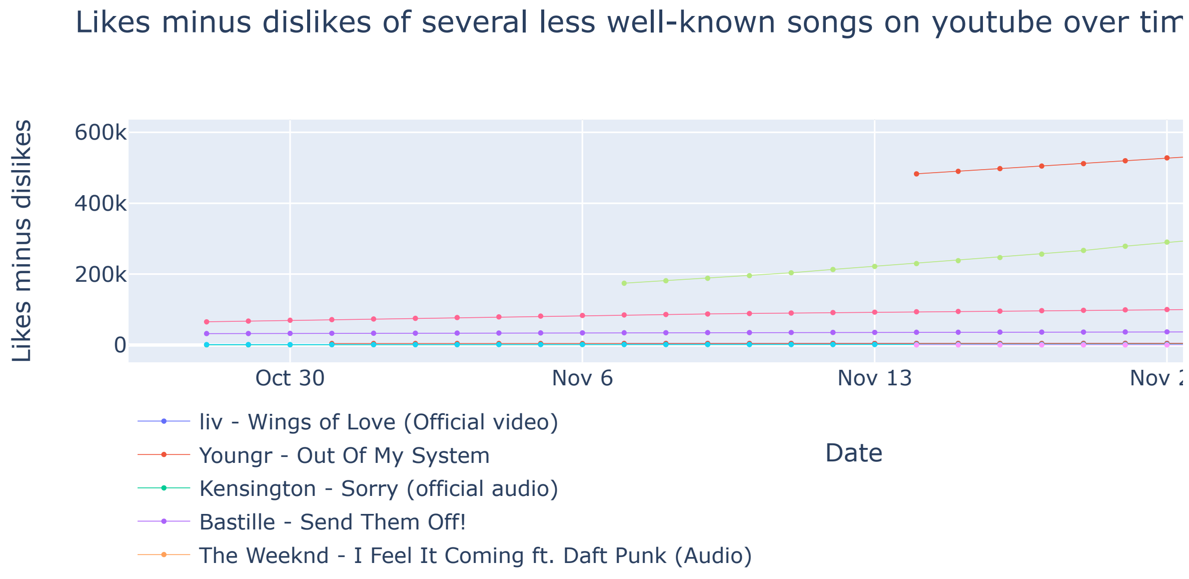
Assignment 1 - Cascading effects

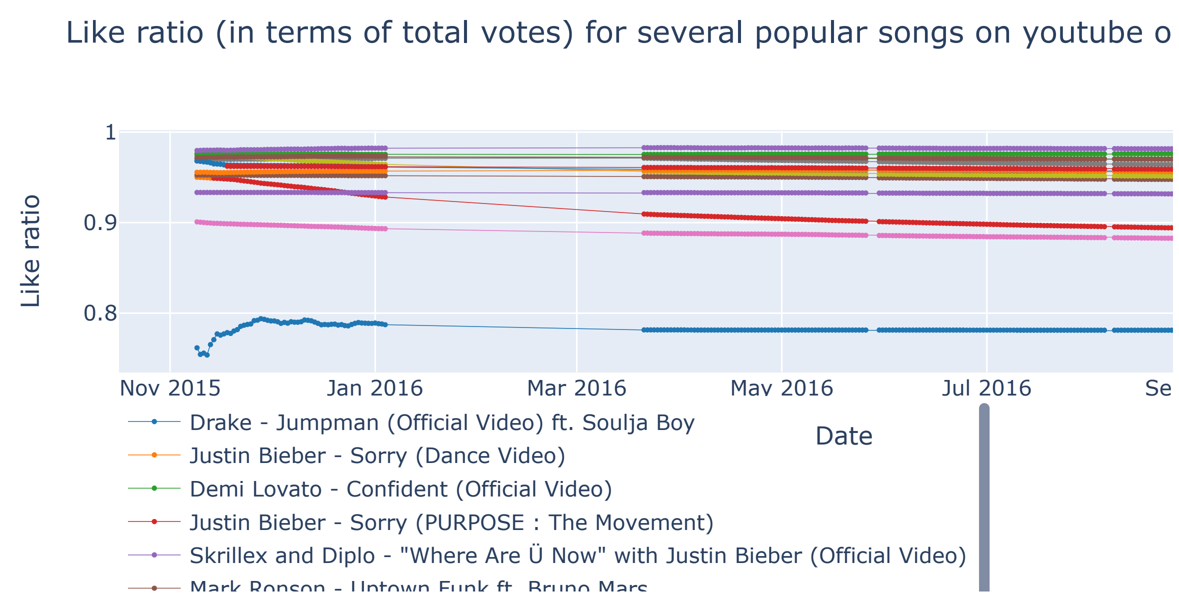
Read the material in Chapter 16 on cascading effects. Plot the difference between the number of likes and dislikes for several songs.

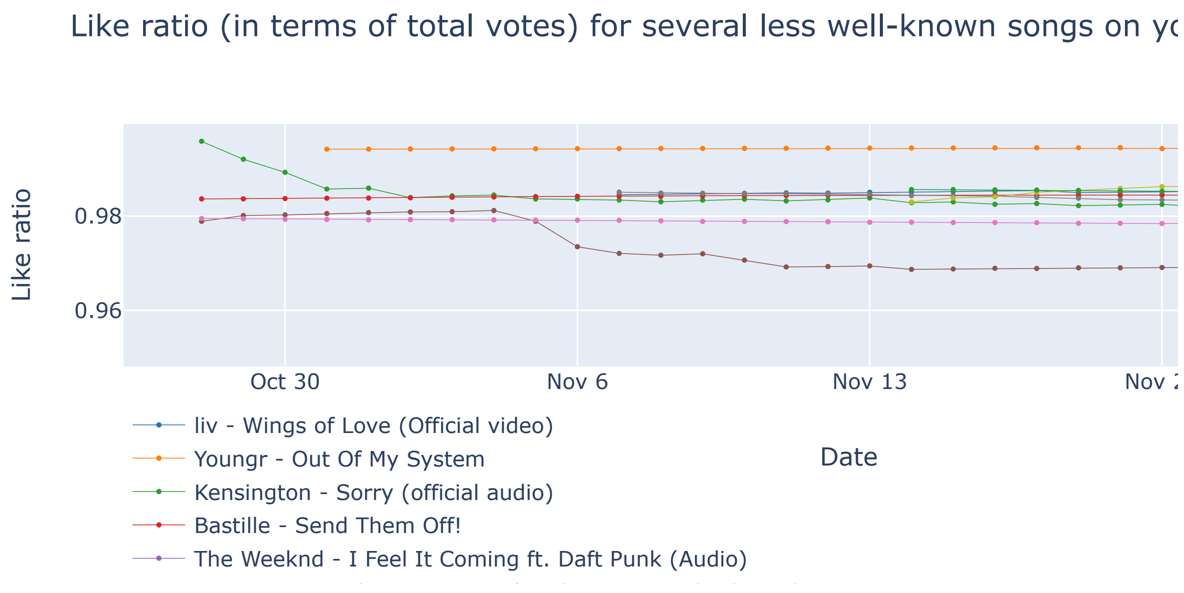
We will plot both the like ratio, $\frac{likes}{likes+dislikes}$, and the difference between the number of likes and dislikes.

In [5]:
```
def find_songs(dataframe,mode='relative_likes'):
    """
    Used to convert a DataFrame containing many songs into a nice dictionary that c
ontains data per songs.
    Useful for when we want to plot data
    """
    dt_songs = {}
    for index, row in sorted(dataframe.iterrows(), key=lambda k: k[1]['timestamp
']): # Sort data by timestamp before doing

# anything to avoid weird errors
        y_id = row['song'] # Get song name

        # For different modes, the data plotted will be different
        if mode=='relative_likes':
            data = int(row['likes'])-int(row['dislikes'])
        elif mode=='like_ratio':
            data = int(row['likes'])/int(row['votes'])
        elif mode=='expectation':
            data = int(row['views'])
        else:
            # Just use the mode directly. For example, using mode='likes' will make
the like count the data
            data = int(row[mode])


        if mode=='expectation': # When plotting expectation (later on), use likes f
or x axis. Otherwise, we always use timestamp
            data2 = int(row['likes'])
        else:
            data2 = row['timestamp']

        if y_id not in dt_songs.keys(): # If this song is not in the dict yet, add
this entry containing 2 lists
            dt_songs[y_id] = [[data],[data2]]
        else: # If the song is already in the dict, just append the data to the exi
sting lists
            dt_songs[y_id][0].append(data)
            dt_songs[y_id][1].append(data2)

    return(dt_songs)



def draw_linechart(figure,dt_data,plot_trendline=False,**kwargs):
    """
    Takes data as generated by the find_songs function, and plots the data per song
    """
    colors=plotly.colors.DEFAULT_PLOTLY_COLORS # List of colors
    i=0
    if plot_trendline:
        # If we plot the trendline, calculate the a list of all data points on the
x-axis for all songs
        all_x = [data[1] for song, data in dt_data.items()]
        flattened_x = [item for sublist in all_x for item in sublist]

    for song, data in dt_data.items():
        color = colors[i] # Use the same color for both trendline and data

        # Set x, y, and name of the data in legend
        kwargs['y'] = data[0]
        kwargs['x'] = data[1]
        kwargs['name']=song
```


Likes minus dislikes of several popular songs on youtube over time
Likes minus dislikes
10M
5M
0
Nov 2015
Jan 2016
Mar 2016
May 2016
Jul 2016
Se
Date
Drake - Jumpman (Official Video) ft. Soulja Boy
Justin Bieber - Sorry (Dance Video)
Demi Lovato - Confident (Official Video)
Justin Bieber - Sorry (PURPOSE : The Movement)
Skrillex and Diplo - "Where Are Ü Now" with Justin Bieber (Official Video)
Mark Ronson - Uptown Funk ft. Bruno Mars


Likes minus dislikes of several less well-known songs on youtube over tim
Likes minus dislikes
600k
400k
200k
0
Oct 30
Nov 6
Nov 13
Nov 2
Date
liv - Wings of Love (Official video)
Youngr - Out Of My System
Kensington - Sorry (official audio)
Bastille - Send Them Off!
The Weeknd - I Feel It Coming ft. Daft Punk (Audio)


Like ratio (in terms of total votes) for several popular songs on youtube o
Like ratio
1
0.9
0.8
Nov 2015
Jan 2016
Mar 2016
May 2016
Jul 2016
Se
Date
Drake - Jumpman (Official Video) ft. Soulja Boy
Justin Bieber - Sorry (Dance Video)
Demi Lovato - Confident (Official Video)
Justin Bieber - Sorry (PURPOSE : The Movement)
Skrillex and Diplo - "Where Are Ü Now" with Justin Bieber (Official Video)
Mark Ronson - Uptown Funk ft. Bruno Mars


Like ratio (in terms of total votes) for several less well-known songs on yo
Like ratio
0.98
0.96
Oct 30
Nov 6
Nov 13
Nov 2
Date
liv - Wings of Love (Official video)
Youngr - Out Of My System
Kensington - Sorry (official audio)
Bastille - Send Them Off!
The Weeknd - I Feel It Coming ft. Daft Punk (Audio)

Do you think we observe cascading effects? Is there a difference between songs that are already popular (in the top-100) and those that are not (megahit or alarmschijf)?.

To answer the question wether a cascading effect is present we instead ask ourselves:
"Is a listeners choice to like or dislike influenced by the previous likes or dislikes?"

We will base our answers mostly on the like ratio, as there seems to be more clear, meaningful trends here, while the difference between likes and dislikes is always a positive, upward trend (which is natural for any song with a like ratio above 0.5, which all of them are).

All of the top100 songs already existed or were already popular at the first point of measurement, so it is difficult to say something about where information cascades start. We are in luck however as the song Jumpman by Drake and Soulja Boy is an accident (we assume) in the dataset. There is a song Jumpman by Drake featuring Future, and a seperate song Jumpman by Souljaboy. The data for the youtube video however has some intersting behaviour. It starts with almost no views and as time progresses the ratio of likes stabilizes. This could be caused by the lack of views after a certain date, thus this is not a good proof for our theory.

The model in Chapter 16 in its pure form cannot be applied to music preferences. Why not? Can you modify the model accordingly? Can you quantify cascading effects in this setting?

We do not know the actual quality of the song, nor do we know how much users like it. We will try to make a model for this, based on the theory presented in the book. This is presented in the table below:

notation	explanation	value
G	song is actually good	P[G] = p
B	song is actually bad	P[B] = 1-p
H	private high signal (listener enjoys)	P[H|G] = q
L	private low signal (listener does not enjoy)	P[L|G] = 1-q
a	high signals of previous people	amount of likes
b	low signals of previous people	amount of dislikes
v_g	payoff when accepting a good state	?
v_b	payoff when rejecting a bad state	?

We have no way of telling objectively wether the sitution is G or B. The songs have no objective quality trait and liking music is very subjective. So, even if we could theoretically try and quantify information cascades using the values in the table above this paragraph, there is no way for us to truly do this based on the data. We also do not know what a good user payoff value would be in such a situation.

The payoff could be an improved or worsened recommendation algorithm, where a user gets more or less songs they actually like.

To try and come to a more general conclusion, we can look at what trends there are in the data. Without cascades, songs would almost immediately get a stable like to dislike ratio, and stay at that ratio. It is a good indicator of a song's quality; if 70% of people would like the song, the like ratio would approach that number as n, the amount of views, increases. While we do not know the statistical distribution for these numbers, is seems reasonable that the amount of views these videos get per day would quickly remove any "noise" in the like ratio.

Thus, variations in the like ratio can then only be attributed to cascading effects: a song is released, it gets some likes (or dislikes), and people who first see the song's video are biased by viewing this positive (or negative) like ratio, thus giving the song a similar "score". However, after a while, the hype for this video is over, and people judge it for its true quality. This can be seen in various songs in the dataset, but is most prominent in Justin Bieber - Sorry, and "DIT IS 4U MET BITTER TASTE".

The sudden dislike of Justin Bieber's song could also be a cascade in and of itself; Justin Bieber is famous for being disliked by internet trolls, and his older music (the song "Baby") is one of the most disliked videos of all time. At this point in time, people find it funny to dislike his songs, which causes other people to do so as well. In any case, this is not a pattern that

Assignment 2 - Network effects

Study the material in Chapter 17. Our goal now is to investigate whether we observe network effects in music preferences. How will you choose the data for this purpose? Which songs will suit most?

The songs that will suit most are ones that we can track their rise and fall of popularity of from the moment they were published. The most suitable candidates are those that have been a 3FM megahit or on the 538 alarmschijf, as they were quite unknown when being published on there. To compare, we will choose some songs from the Youtube and Spotify top 100 list. For these songs, the popularity / viewcount is the most interesting data to plot over time.

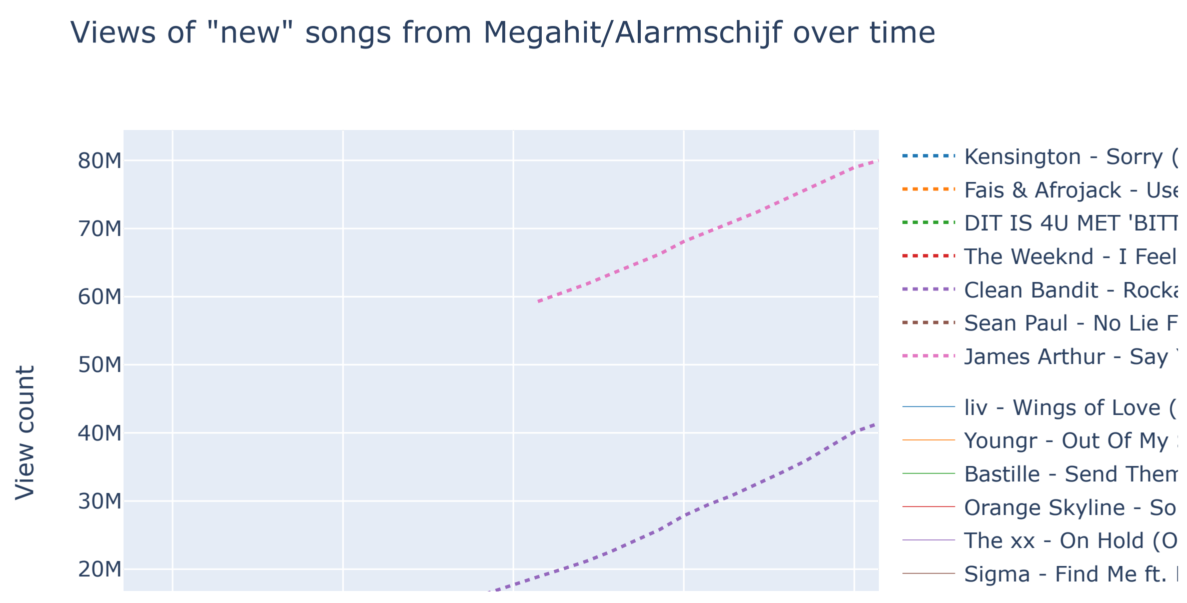
For several songs of your choice plot the actual number of views against time. Assume that without network effects we can expect that users visit a website with a certain frequency and view the song if it matches their taste. Hence, the expected number of views grows linearly in time. Compare your plot to Figure 17.4. Do you think you observe network effects?

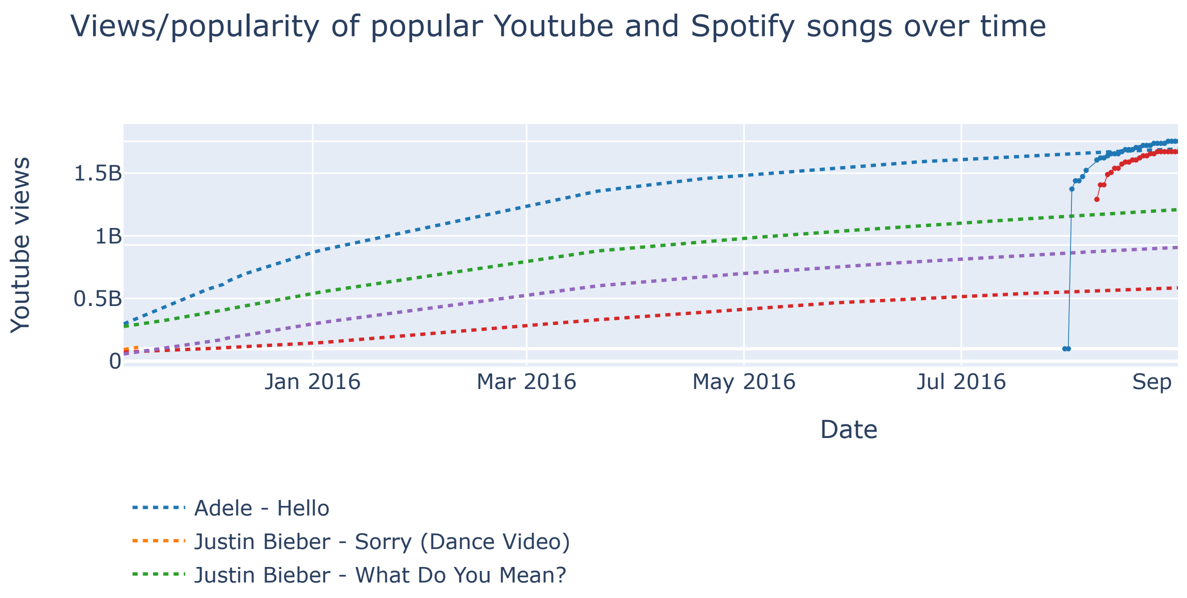
We plot the viewcount of the Megahit / Alarmschijf songs in the first plot, and the viewcount + popularity of some random top 100 YouTube / Spotify songs in the second plot.

In the first plot, Alarmschijf songs have a dotted line, while Megahit songs have an opaque line.

In the second plot, the Youtube songs have a dashed line. Their y-axis is the view count, as seen on the left side of the plot. The Spotify songs have an opaque line, and their y-axis is "popularity" (a number calculated using some algorithm, rannging from 0 to 100), as seen on the right side of the plot.

In [6]:
```
# Grab the data
some_youtube_songs = data_youtube100[data_youtube100.song.isin(data_youtube100.son
g.head(n=5))]
some_spotify_songs = data_spotify100[data_spotify100.song.isin(data_spotify100.son
g.head(n=5))]

# Make figures
unknown_songs_figure = go.Figure()
youtube_songs_figure = go.Figure()

# Plot alarmschijf data
alarmschijf_data = find_songs(data_alarmschijf,mode='views')
draw_linechart(unknown_songs_figure,alarmschijf_data,mode='lines',line={'dash':'dot
'},legendgroup='alarmschijf')

# Plot megahit data
megahit_data = find_songs(data_megahit,mode='views')
draw_linechart(unknown_songs_figure,megahit_data,mode='lines',legendgroup='megahit
')

# Plot youtube top100 data
some_youtube_songs_data = find_songs(some_youtube_songs,mode='views')
draw_linechart(youtube_songs_figure,some_youtube_songs_data,mode='lines',line={'das
h':'dot'},legendgroup='some_youtube_songs')

# Plot spotify top100 data
some_spotify_songs_data = find_songs(some_spotify_songs,mode='popularity')
draw_linechart(youtube_songs_figure,some_spotify_songs_data,plot_trendline=False,ya
xis='y2',legendgroup='some_spotify_songs')


### Update plot titles and such

unknown_songs_figure.update_layout(
    title='Views of "new" songs from Megahit/Alarmschijf over time',
    xaxis_title='Date',
    yaxis_title='View count')

youtube_songs_figure.update_layout(
    title='Views/popularity of popular Youtube and Spotify songs over time',
    xaxis_title='Date',
    yaxis=dict(title="Youtube views")

    ,yaxis2=dict(
        title="Spotify popularity",
        anchor="free",
        overlaying="y",
        side="right",
        position=1
    ),
    legend=dict(x=0, y=-2)
)

# Show plots

unknown_songs_figure.show()
youtube_songs_figure.show()
```


Views of "new" songs from Megahit/Alarmschijf over time
View count
80M
70M
60M
50M
40M
30M
20M
Kensington - Sorry (
Fais & Afrojack - Use
DIT IS 4U MET 'BITT
The Weeknd - I Feel
Clean Bandit - Rocka
Sean Paul - No Lie F
James Arthur - Say
liv - Wings of Love (
Youngr - Out Of My
Bastille - Send Them
Orange Skyline - So
The xx - On Hold (O
Sigma - Find Me ft.


Views/popularity of popular Youtube and Spotify songs over time
Youtube views
1.5B
1B
0.5B
0
Jan 2016
Mar 2016
May 2016
Jul 2016
Sep
Date
Adele - Hello
Justin Bieber - Sorry (Dance Video)
Justin Bieber - What Do You Mean?

Plot explanation

The 3FM Megahit and 538 Alarmschijf songs (first plot) all seem to have quite a linear relationship between time and popularity, while the more popular songs from the top 100 (second plot) have a relationship between time and popularity which is very similar to figure 17.4 (a fast rise at first, then a more slow stabilization towards the equlibrium). The data from Spotify seems to be much more extreme in this than the data from YouTube. We do not know what causes this, as Spotify does not release the way that they calculate this "popularity" ranking, which is a number from 0 to 100, to the public. It may simply be the percentile the song is in, when sorting all songs by the amount of plays in one week. All we know is that it is not updated in real-time, and may take a few days to be updated (https://community.spotify.com/t5/Content-Questions/Artist-popularity/m-p/4734566/highlight/true#M32231).

Thus, we can see quite clearly that network effects are not visible for impopular songs (users simply come across it by visiting a platform and clicking songs that match their preferences, causing a linear growth in views), while they are visible for more popular songs (people listen to them because of their popularity, causing a growth like in figure 17.4).

Assignment 2 (continued)

How will you interpret the network benefit function f, the intrinsic interest function r, and the price p? Try to specify the model that best fits the data.*

For plots like for functions f and r, normally the share of consumers that buys something is plotted on the x axis, and the y-axis represents the "price" of the good. There are multiple interpretations possible for this, and we will explore all of them.

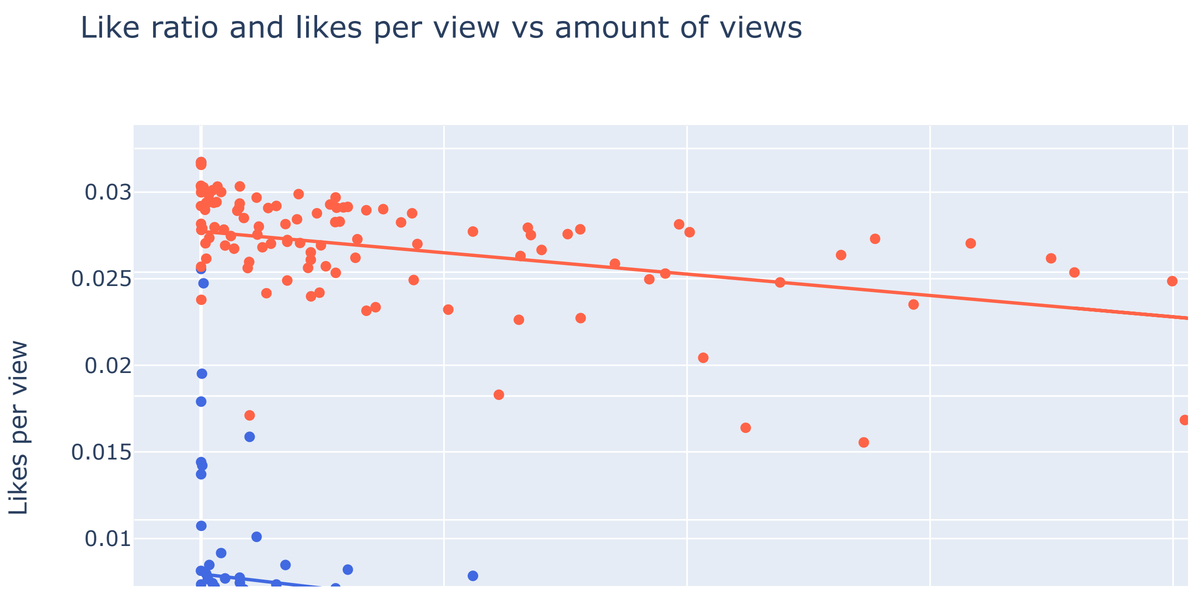
In the first case, the share of consumers can be represented by the amount of views (amount of people that took the time to click on the video). Then, what do we plot on the y-axis? The amount of likes is not interesting to put on the y-axis, as there is a very strong (yet uninteresting and trivial) correlation between the amount of likes and the amount of views, which will tell us nothing about network benefits or intrinsic interest. We chose to plot two metrics for the price: the like ratio, $\frac{likes}{total\ votes}$ and the likes per view. Both are good estimators for the true price, as they show the share of people who think that paying the price of clicking the video was worth it. This can be seen in the first plot. The likes per view and like ratio use separate y axes because, while they use the same data for the x-axis, the scale of their y-axis is completely different. The y-axis for likes per view can be seen on the left side, and the y-axis for like ratio can be seen on the right side.

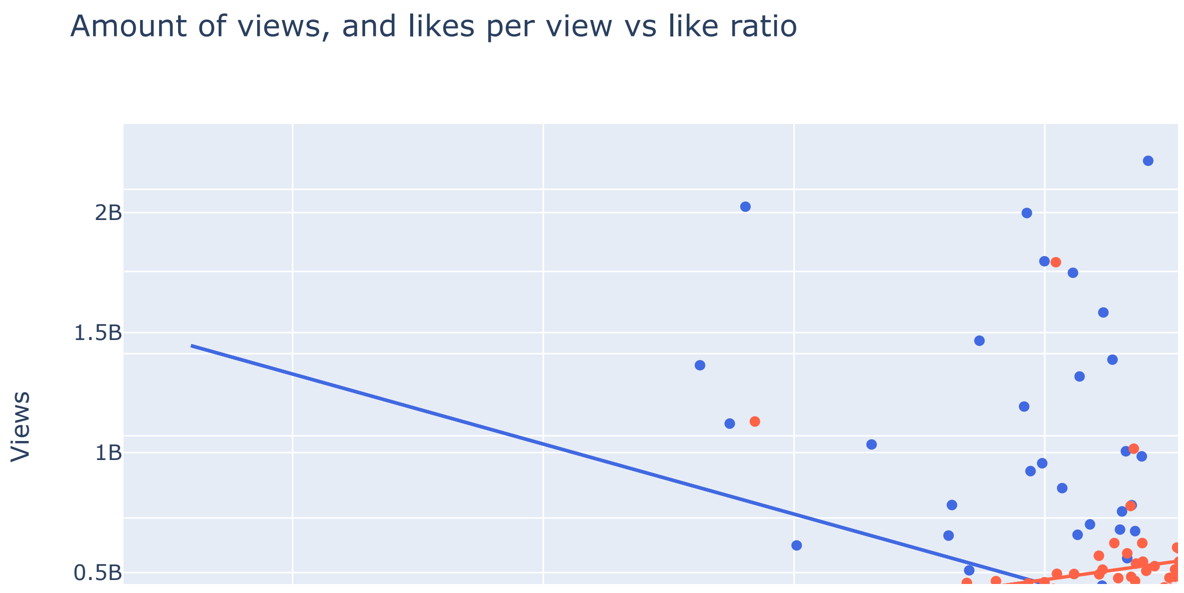
For the second case, the share of consumers willing to "pay" for the music can be represented by the like ratio. If people dislike a song, they didn't think that spending the time to click on it and start listening is worth it. Therefore, the like ratio is the true share of people that think the price of clicking on this song is worth it. Then how do we model the price? In this case, there are two possibilities. The first one is the amount of views being a way to represent the price: that amount of people were willing to pay the true price of clicking the video. We think that this approach dos not make any sense in retrospect, but the plot was left in to show the correlation. The y-axis for this is on the left side of hte plot. Another approach would be to plot the likes per view on the y-axis. This represents the engagement rate of the video. The y-axis for this is on the right side of the plot.

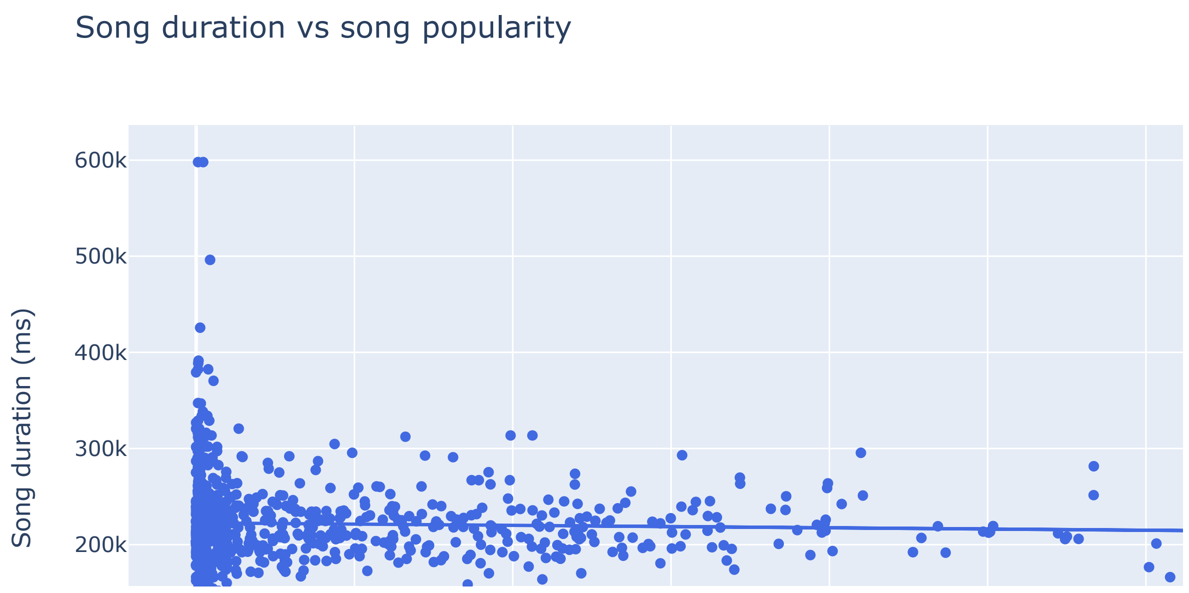
We think that the first case is more likely to be correct, as it is quite hard to make good arguments for the second case. We only discovered this after already plotting this data during the exploratory phase, and left it in for the sake of showing our work.

A third possible way to plot the data uses Spotify data. Here, the song's max popularity over all time is the share of the population willing to "buy" the product, while the "price" they pay is the duration of the song in milliseconds. This is seen in the third plot.

In [7]:
```
# Gathering data

merged_data = pd.concat([data_youtube100,data_alarmschijf,data_megahit]) #concatena
te youtube 100, alarmschijf, and megahit data
data = merged_data[merged_data.timestamp == u'2016-11-28'] # Plot data on november
28 (last date at which data was collected for all)

dislikes_over_votes = (data.dislikes/data.votes)
likes_over_votes = data.likes/data.votes
likes_over_dislikes = data.likes/data.dislikes

### Plot 1
figure = go.Figure()

# Plot likes per view and trendline
figure.add_trace(go.Scatter(x=data.views,y=data.likes/data.views,mode='markers',nam
e='Likes per view',marker={'color':'royalblue'}))
trendline1 = np.poly1d(np.polyfit(data.views,data.likes/data.views,1))
figure.add_trace(go.Scatter(x=data.views,y=trendline1(data.views),mode='lines',nam
e='Likes per view trendline',line={'color':'royalblue'}))

# Plot like ratio and trendline
figure.add_trace(go.Scatter(x=data.views,y=likes_over_votes,mode='markers',name='Li
ke ratio',yaxis='y2',marker={'color':'tomato'}))
trendline2 = np.poly1d(np.polyfit(data.views,likes_over_votes,1))
figure.add_trace(go.Scatter(x=data.views,y=trendline2(data.views),mode='lines',nam
e='Like ratio trendline',line={'color':'tomato'},yaxis='y2'))

### Plot 2
figure2 = go.Figure()

# Plot views
figure2.add_trace(go.Scatter(x=likes_over_votes,y=data.views,mode='markers',name='V
iews',marker={'color':'royalblue'}))
trendline1 = np.poly1d(np.polyfit(likes_over_votes,data.views,1))
figure2.add_trace(go.Scatter(x=likes_over_votes,y=trendline1(likes_over_votes),mod
e='lines',name='View count trendline',line={'color':'royalblue'}))

# Plot likes per view
figure2.add_trace(go.Scatter(x=likes_over_votes,y=data.likes/data.views,mode='marke
rs',name='Likes per view',yaxis='y2',marker={'color':'tomato'}))
trendline2 = np.poly1d(np.polyfit(likes_over_votes,np.array(data.likes/data.views),
1))
figure2.add_trace(go.Scatter(x=likes_over_votes,y=trendline2(likes_over_votes),mod
e='lines',name='Likes per view trendline',yaxis='y2',line={'color':'tomato'}))

### Plot 3
figure3 = go.Figure()
# Plotting duration against sum of all instances of song's popularity (discrete ver
sion of integral of popularity over time)

# Get new dataset, from spotify
data = get_n_songs(data_spotify100,n=None)

# Get the max popularity of each song
max_popularities=( [ (data[data.spotify_id == spotify_id].duration_ms.iloc[0] , sum
(list(data[data.spotify_id == spotify_id].popularity)) ) for spotify_id in data.spo
tify_id.unique()])

# Make this into two lists, for plotting purposes
ls_durations, ls_max_popularities = map(list, zip(*max_popularities))

#Only use data from last date at which data was collected
data = data[data.timestamp == u'2015-12-15']
```


Like ratio and likes per view vs amount of views
Likes per view
0.03
0.025
0.02
0.015
0.01


Amount of views, and likes per view vs like ratio
Views
2B
1.5B
1B
0.5B


Song duration vs song popularity
Song duration (ms)
600k
500k
400k
300k
200k

As we can see in plot 1, there is an obvious negative correlation between the amount of views, and the like ratio or likes per view. This gives us a figure similar to figure 17.2. More popular videos seem to lead to less user interaction, and attract more dislikes (which could be due to internet trolls, haters, and people disliking something as it becomes too popular). Because the amount of likes per view does not suffer from this problem, we think this is the most interesting way to model the price in a price-consumersplot. When the amount of people that have "consumed" media of high quality is low, they are likely to want to spread it to friends, and press the like button. Thus, they think the 'price' of the video was too low, and more people should click the video.

It is less easy to obtain meaningful conclusions from the second plot. For the amount of likes, we can come to the same conclusion as we did one paragraph above. The observation that there is a positive relationship between like ratio and likes per view is interesting, but this is not a proper model for customer interest or network effects.

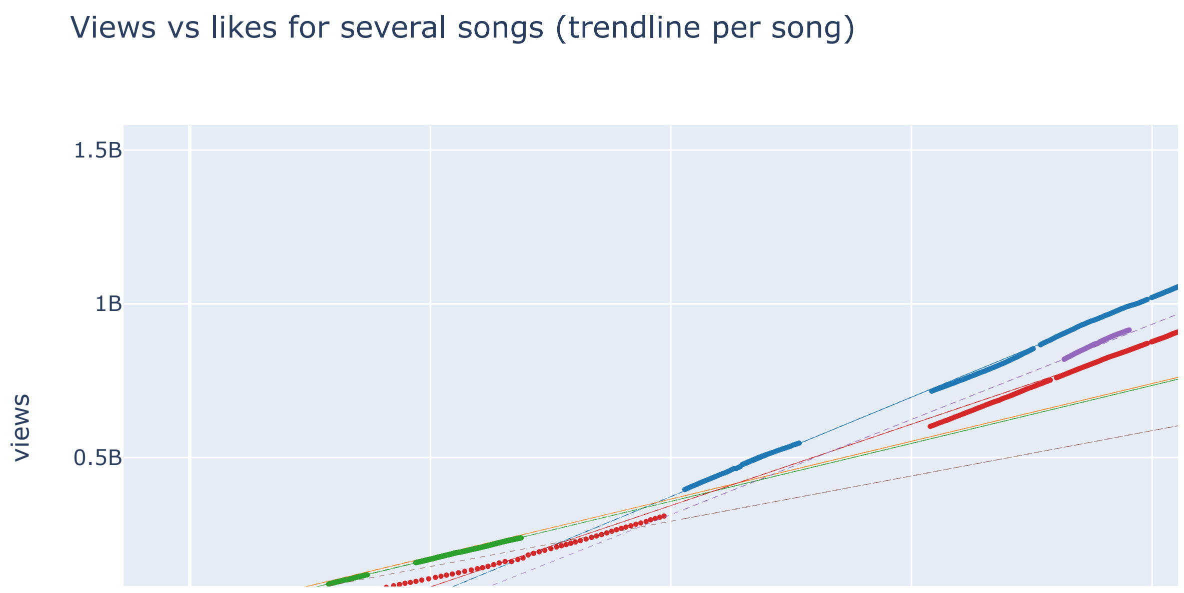
The third plot is also not as helpful as we hoped. It is obvious that songs cluster around being 210k ms in length, and thus the most popular songs are also around this. However, song popularity has no influence on the duration (which makes sense, as duration is an independent variable which can be changed by the artist, while popularity is not). This plot would have been better with the x-axis and the y-axis swapped, but that could not have been a potential model for price vs. consumer interest.

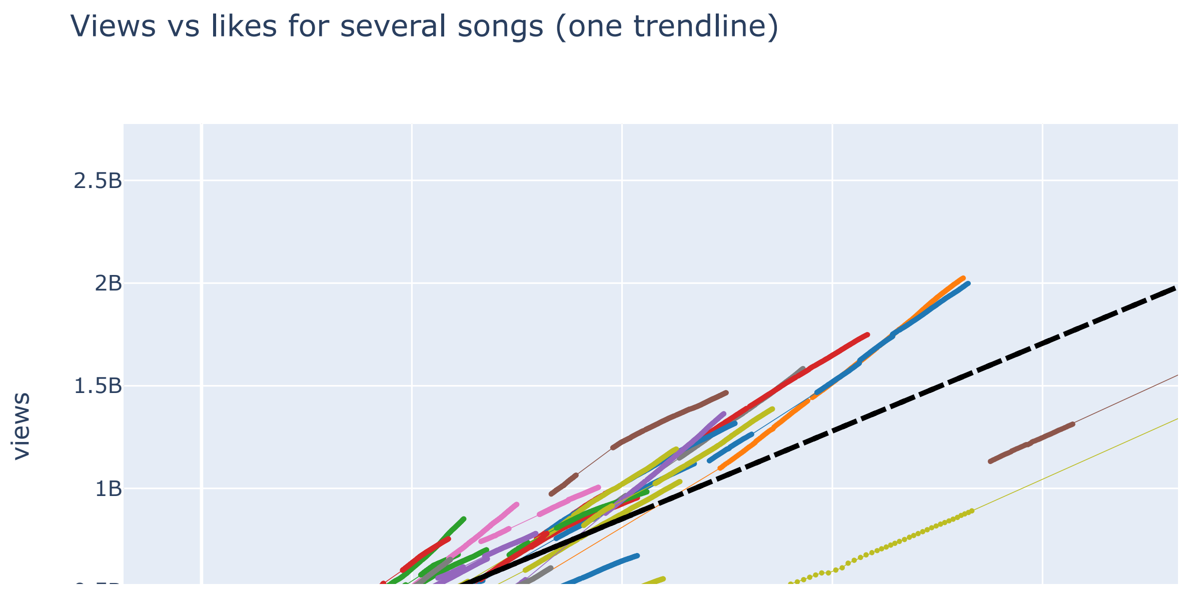
Thus, we were able to find a suitable model for r(x): the likes per view vs the view count. The model for the price is then: the amount of likes per view (which could be seen as an approximation of a video's "true quality", therefore being directly proportional to the "price" people are willing to pay).

We were unable to find a suitable model for f(x) (the benefit of the good when x amount of the population use it). This is supposed to have a positive correlation, and the only one we found so far is the correlation between the like ratio and the amount of likes per view. But this does not seem like a good approach (the amount of likes per view has a trivial correlation with the like ratio). We will continue looking for a way to model this below.

We will plot the amount of views against the amount of likes, hoping to find something like figure 17.9 (outcome vs expectations). We will use two plots: one with a trendline per song, and one with only one trendline, for all songs.

In [8]:
```
songs=get_n_songs(data_youtube100,n=6,mode='sample') # Get n random songs

data1 = find_songs(songs,mode='expectation') # X axis will be likes, Y axis will be
views
figure = go.Figure()
draw_linechart(figure,data1,plot_trendline=True,mode='markers') # plot this data, a
nd the trendline for each song
figure.update_layout(
    showlegend=False,
    title='Views vs likes for several songs (trendline per song)',
    xaxis_title='Likes',
    yaxis_title='views'
)
figure.show()

songs2=get_n_songs(data_youtube100,n=None) # We repeat this for a much larger n

songs_2 = get_n_songs(data_youtube100,n=None)
data2 = find_songs(songs_2,mode='expectation')
figure2 = go.Figure()
draw_linechart(figure2,data2) # but do not plot the trendline per song...
trendline2 = np.poly1d(np.polyfit(data_youtube100.likes,data_youtube100.views,1)) #
...getting one from all songs instead.
figure2.add_trace(go.Scatter(x=songs_2.likes,y=trendline2(songs_2.likes),line={'col
or':'black','dash':'dash','width':3},name='Population trend line'))
figure2.update_layout(showlegend=False,
    title='Views vs likes for several songs (one trendline)',
    xaxis_title='Likes',
    yaxis_title='views')
figure2.show()
```


Views vs likes for several songs (trendline per song)
1.5B
1B
0.5B
views


Views vs likes for several songs (one trendline)
2.5B
2B
1.5B
1B
views

Explanation of plots

The second plot is interesting to look at, but does not show the behavior we wanted to see. The first one does show the behavior we expected: the shared expectation (amount of likes) influences the realization (view count). If the data ever gets too far above or below the trend line (which represents the video's average views per like, its intrinsic quality), there will be a correction, and the data will approach the trend lineagain. This behavior is similar to the behavior in figure 17.9, except for the fact that our expectation and realization can only move upwards.

This can also be seen as a model for f(x): if the amount of people who view the song (amount of consumers) goes up, the amount of people who like it (benefit of the good due to network effects) goes up, and vice versa. This would make more sense with a plot that has views on the x-axis and likes on the y-axis, but the correlation can be seen clearly in this plot as well. The current axis distribution was chosen to find a realization-expectation plot, where the amount of likes is the expectation. This choice made sense for what we were trying to find, but is not ideal for finding f(x).

TODO @Emiel jij zei iets over: slope van trendline is hoeveel mensen willing zijn te kopen? Kun je dat hier uitleggen?

Assignment 3 - Popularity effects

include at least:

- plot
- code explanation

Study the material in Chapter 18. We investigate whether rich-get-richer phenomenon explains the dynamics of the number of views.

Plot the distribution of the number of views among the songs on several different days. Do you observe power laws?

For this, we will add show 3 plots, with youtube data.

We start with a linear plot of the data, with the trendline from the lin-log plot.

Next we show a log-lin plot of the data, with a trendline calculated. Straight lines in log-lin plots show exponential relations. We also added the trendline from the log-log plot (dashed) to illustrate the difference. (We do not show the trendline created from the log-log plot in the linear plot, because of the enormous difference in scale this causes)

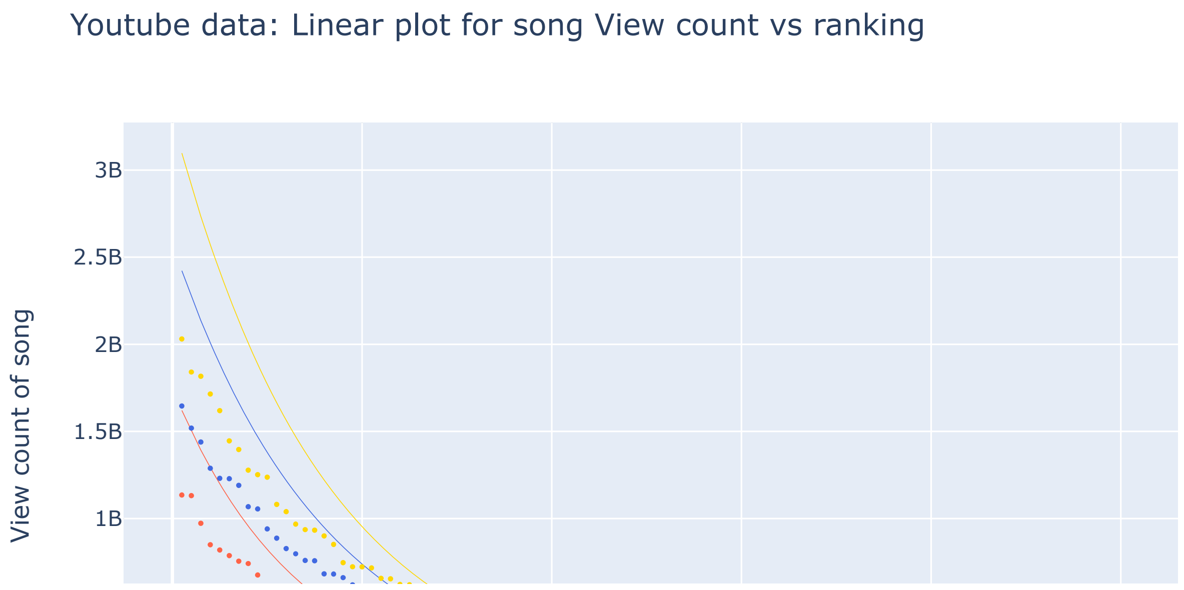
Finally, we show a log-log plot of the data, with a trendline calculated for it (dashed line). Straight lines in log-log plots show power law relationships. We also added the trendline from the lin-log plot to illustrate the difference (non-dashed).

We then repeat this process for Spotify data to see if there are any interesting differences.

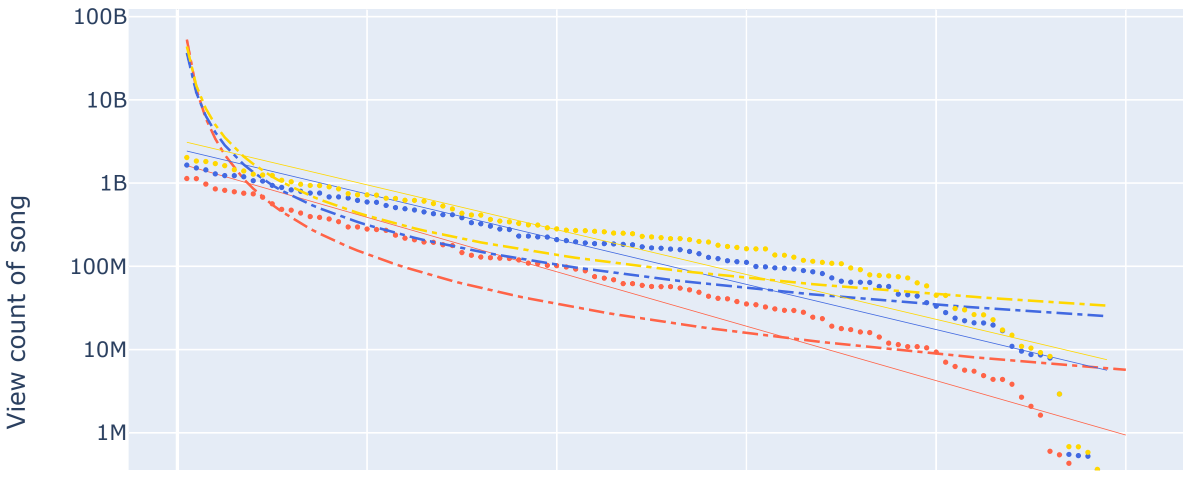
In [9]:
```
dates = [u'2015-11-09', u'2016-04-13',u'2016-09-16'] # list of dates that we will u
se for plotting
colors=['tomato','royalblue','gold'] # List of colors that we will use

all_data = [data_youtube100,data_spotify100] #List of data sets that we will use
all_data_names = ['Youtube data', 'Spotify data'] # Names of those sets
metrics = ['View count', 'Popularity'] # Metrics used for those sets
for i in range(len(all_data)): # Loop through data sets

    #Set variables
    dataset= all_data[i]
    dataset_name = all_data_names[i]
    metric = metrics[i]

    print '\n\n' + dataset_name + '\n\n' # Print that we are working on this data s
et

    # Prepare figures, called day1_fig because they were originally intended do be
used for only one day
    day1_fig = go.Figure()
    day1_fig_log = go.Figure() # linlog plot
    day1_fig_loglog = go.Figure() #loglog plot
    for date in dates: # Loop through the dates
        try:
            # If spotify data: data is the view count
            data = sorted(dataset[dataset.timestamp == date].views,reverse=True)
        except:
            # if spotify data: data is the popularity
            data = np.array(sorted(dataset[dataset.timestamp == date].popularity,re
verse=True))+1 # add 1 to avoid divide by zero errors in popularity

        color = colors[dates.index(date)] # Set color for plot
        x = np.array(range(len(data))) + 1 # Array with same size as data, starting
from 1 to avoid pesky divide-by-zero errors

        # Calculate trend lines (and their residuals)
        polynomial_coefficients_log, residuals_log = np.polyfit(x,np.log(data), 1,f
ull=True)[:2] # Returns coefficients of a polynomial of degree 1 (just a linear rel
ation) with least square fit to data
        polynomial_coefficients_loglog, residuals_loglog = np.polyfit(np.log(x),np.
log(data), 1,full=True)[:2] # Returns coefficients of a polynomial of degree 1 (jus
t a linear relation) with least square fit to data

        print 'Sum of the squares of the fit errors for ' + date + ' lin-log trendl
ine: ' + str(round(residuals_log[0],2))
        print 'Sum of the squares of the fit errors for ' + date + ' log-log trendl
ine: ' + str(round(residuals_loglog[0],2))

        # Give these trend lines human-readable names
        polynomial_name_log = 'Exponential fit: e^(' + str(round(polynomial_coeffic
ients_log[0],2)) + 'x) + e^' + str(round(polynomial_coefficients_log[1],2))
        polynomial_name_loglog = 'Power law fit: e^' + str(round(polynomial_coeffic
ients_loglog[1],2)) + ' * x^' + str(round(polynomial_coefficients_loglog[0],2))

        # Actually create polynomials from these
        polynomial_log = np.poly1d(polynomial_coefficients_log) # Generate a polyno
mial from these coefficients
        polynomial_loglog = np.poly1d(polynomial_coefficients_loglog) # Generate a
polynomial from these coefficients

        date_name = dataset_name + ' from ' + date # Human-readable name used for d
ata in legend

        # Plot all the data!
```

```
Youtube data
```

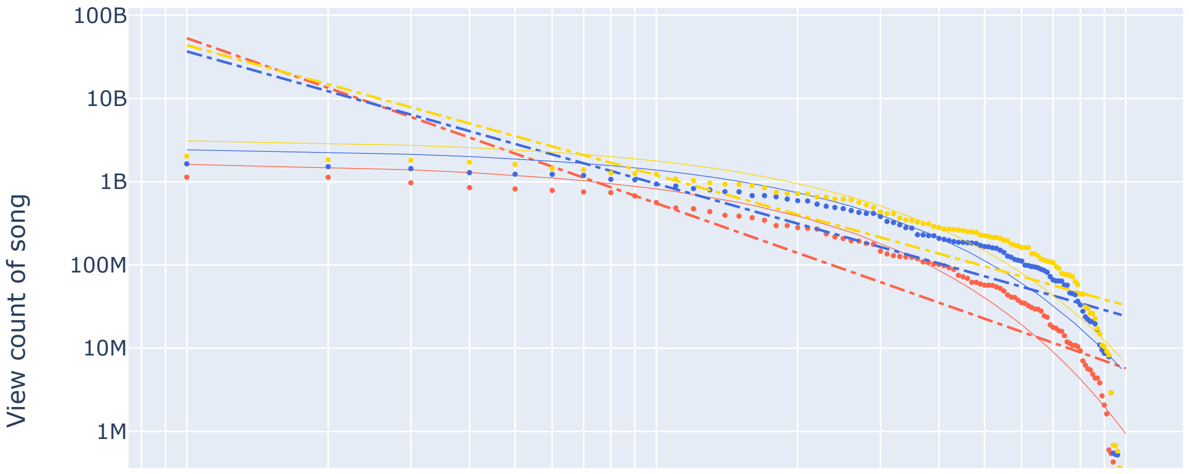
```
Sum of the squares of the fit errors for 2015-11-09 lin-log trendline: 61.25
Sum of the squares of the fit errors for 2015-11-09 log-log trendline: 197.94
Sum of the squares of the fit errors for 2016-04-13 lin-log trendline: 71.72
Sum of the squares of the fit errors for 2016-04-13 log-log trendline: 167.16
Sum of the squares of the fit errors for 2016-09-16 lin-log trendline: 80.17
Sum of the squares of the fit errors for 2016-09-16 log-log trendline: 177.81
```



Youtube data: Log-lin plot for song View count vs ranking

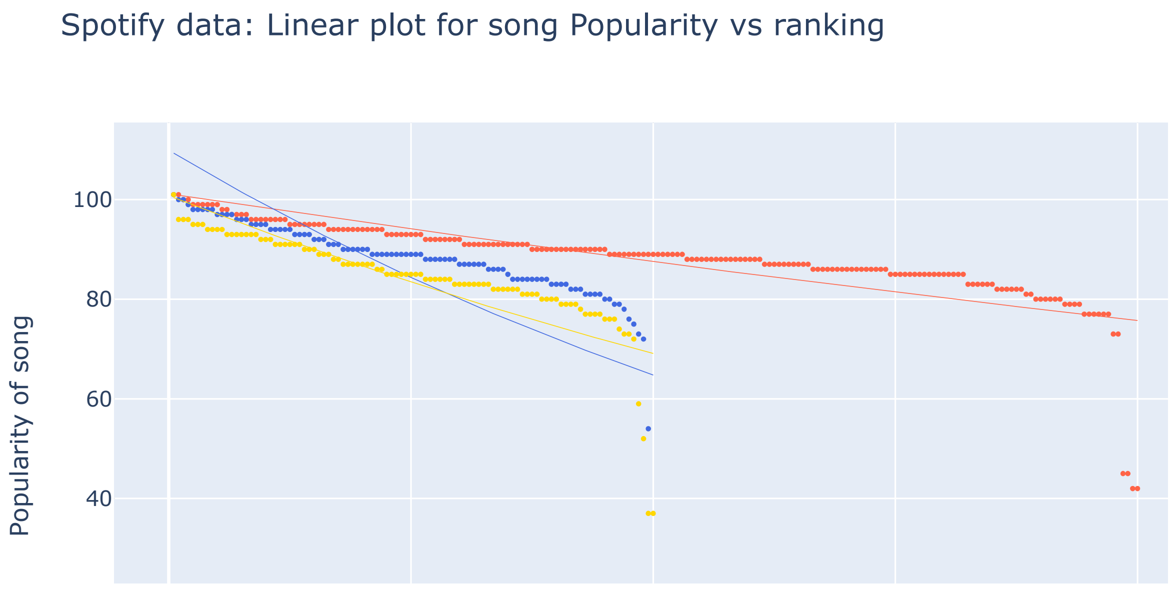


Youtube data: Log-log plot for song View count vs ranking

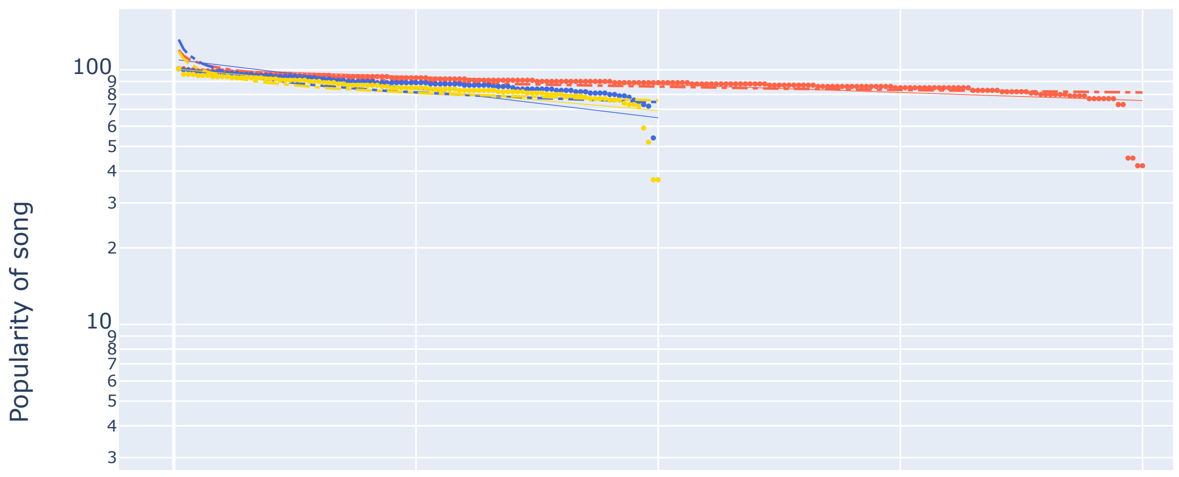


```
Spotify data
```

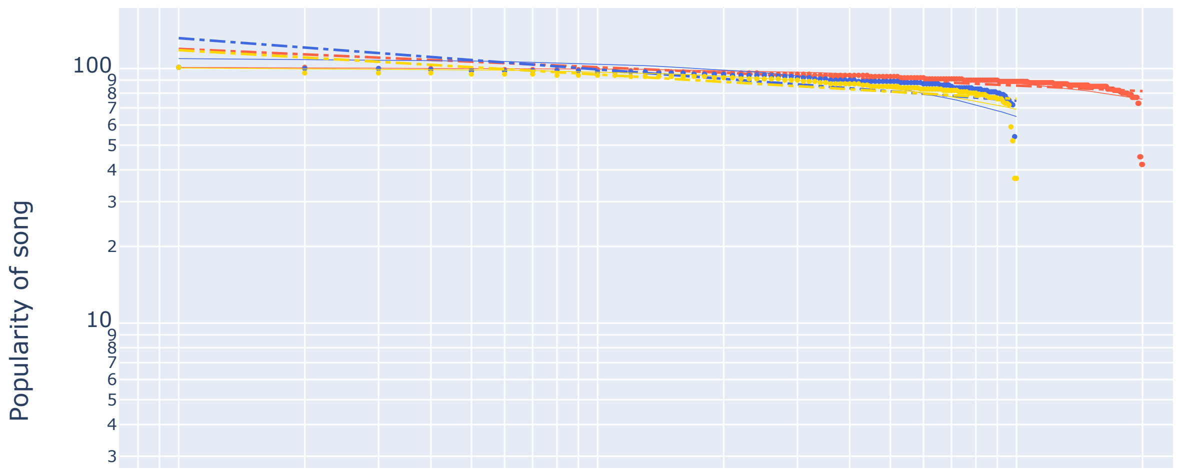
```
Sum of the squares of the fit errors for 2015-11-09 lin-log trendline: 1.4
Sum of the squares of the fit errors for 2015-11-09 log-log trendline: 1.85
Sum of the squares of the fit errors for 2016-04-13 lin-log trendline: 18.27
Sum of the squares of the fit errors for 2016-04-13 log-log trendline: 19.31
Sum of the squares of the fit errors for 2016-09-16 lin-log trendline: 1.05
Sum of the squares of the fit errors for 2016-09-16 log-log trendline: 1.44
```



Spotify data: Log-lin plot for song Popularity vs ranking



Spotify data: Log-log plot for song Popularity vs ranking



Explanation of plots:

The fit of the lin-log plot seems to be better than the fit on the log-log plot. This, combined with the large amount of outliers for the trendlines on the log-log plot, seems to suggest that the distribution might be an exponential one (of the form $b + a^{cx}$) instead of a power law (of the form $b \cdot x^{-c}$). The mean squared error (printed above the plots) for the lin-log trendline is far lower than the error for the log-log trendline.

While we believe that an exponential relationship is more likely than a power law, one can also see that the trendline for a power-law is is a barely acceptable fit for the data, with c ranging between 1.5 and 2 (quite a weak power law).

Doing the same for the Spotify data was an afterthought, as the exercise mentioned 'views', thus suggesting we should only do this for YouTube data. It is ovious that the trend here is linear; the lin-log trendline uses a very small slope which creates a linear fit. This is quite interesting. The way Spotify categorizes their data seems to be made to compensate for the effects of power laws, so that popularity is linearly distributed accross the songs. This may be done to create a more intuitive, huuman-readable number instead of the enormous view counts of popular YouTube videos, which are hard to imagine.

Exercise 3 (continued)

Assume that the number of views in the next day is proportional to the total number of views up to the day before. Argue that in this case the number of views will grow exponentially in time.

This is quite easy to prove. Let's say the amount of views of song i, v_i, is multiplied by some constant x_i for that song, each day.

After t days, the amount of views is:

$$v_i(t) = v_{i_{initial}} \cdot {x_i}^t$$

There is an obvious exponential correlation between time and the number of views.

Look again at the plots for the number of views over time that you produced in assignment 2. Do we observe exponential growth on data? Maybe we observe exponential growth at least some periods of time? Do you think we observe the rich get richer phenomenon?

We do not observe exponential growth in our data. Instead, we see trends that resemble linear growth (for songs that are not popular) or logarithmic growth (for songs that are popular). Popular songs that are just entering the mainstream grow really quickly for a short time, sometimes doubling their popularity overnight. This could definitely be seen as exponential growth for a short time, with $x_i = 2$.

If the amount of people listening to (Western, English-language) songs were to be infinite, we might see exponential growth. However, because the cap of the population is reached quite quickly, we do not observe exponential growth for long periods of time.

We can observe the rich get richer phenomenon easily: popular songs (in the top 100) have quick growth, which stabilizes after a while. New, non-popular songs (alarmschijf, megahit) have slower, linear growth (unless they were already popular).

Compare the ranking of songs in the Spotify data set (based on their position in the top-100) with the ranking of songs in the YouTube data set (based on their number of views) over time. Are the outcomes in line? Why (not)?

We already did this before (in the previous cell, where we explain the plots in which we try to find power laws, and in exercise 2, where we comment on the growth over time). However, we will quickly summarize our findings here.

View counts for YouTube have a clear exponential or power law distribution, thus the plots are highly skewed. A small percentage of videos hold a very large share of the views. If a video is unpopular, its growth over time is linear, but if a video is popular we will see a very fast growth which then subsides (like a logarithmic function).

Spotify popularity is linearly distributed. The most popular song has a ranking of 100, and the 200 songs after that are mostly distributed in a popularity range of 80-100. Thus, there are no power laws here. Growth over time for Spotify popularity seems to always be quite fast, and then slow, like a logarithmic function. This is similar to the correlation seen in popular YouTube videos, but it is much faster here. We do not have any data for unpopular Spotify songs to compare this to, and we do not know how Spotify calculates their popularity ranking, thus drawing meaningful conclusions from this is difficult. However, it is safe to say that the outcomes are not in line, as the metrics are totally different.

Assignment 4 - Information diffusion

Read Chapter 19. Do you think the model of information diffusion applies for music preferences? How can you observe this on the data? Is it related to other phenomena discussed above? Again, there is no one right answer to these questions, try to formulate your own ideas.

It is likely that information diffusion applies to listening music. However, probably much less clear than, for example, adopting a technology that gives a large benefit when your connections use it as well. Factors that may make the effect less pronounced are:

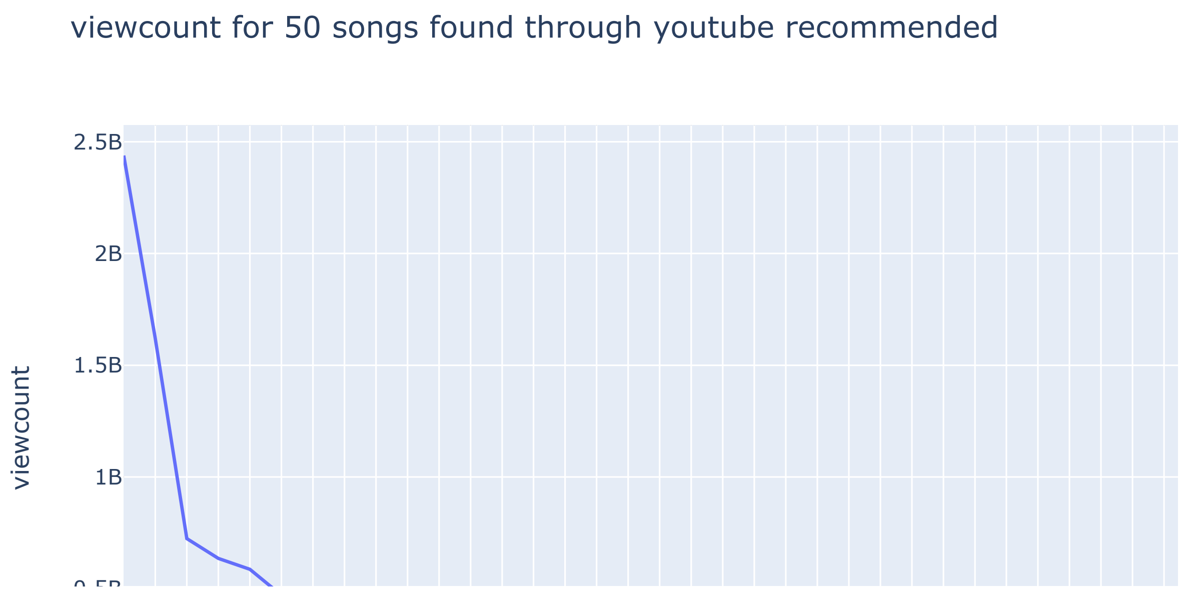
- Music-preference comes from a personal taste
- Radio stations and popular playlists prescribe what will be popular
- There is a whole lot of music out there

On the other hand:

- For trends in music the effect may become clearer when looking at artists or (sub)genres isntead of single songs.
- A preference in music might change, this can be influenced by a persons environemt.
- Music quickly spreads through a network, if your friends like a song they might recommend it to you.
- Music is a common conversational topic, so listening to certain artists that your network listens to gives you a benefit of fitting in.
- It occurs fairly frequently that clusters or tightly knit groups listen to the same genre of music or start liking the same genre over time.

Assignment 5 - Create your own data

In [10]:
```
api_key = "AIzaSyAWckP0mk930l76QTympE1IYWHD6snodB8"
from apiclient.discovery import build
youtube = build('youtube', 'v3', developerKey = api_key)

origin_id = "34Na4j8AVgA" # weeknd - starboy ft. daftpunk
ls_recommended_songs = ['34Na4j8AVgA', 'P_SlAzsXa7E', '52Gg9CqhbP8','6okxuiiHx2w','
ebILIKHi9wo','pBkHHoOIIn8','PT2_F-1esPk',
                        'a5uQMwRMHcs','Kp7eSUU9oy8','qCTMq7xvdXU','LlU4FuIJT2k','tt
WQK5VXskA','CxnaPa8ohmM','K3Qzzggn--s','yvHYWD29ZNY','LdyabrdFMC8',
                        'qFLhGq0060w','CfihYWRWRTQ','Ic5vxw3eijY','FYH8DsU2WCk','mO
KMoL8_jaQ','LdyabrdFMC8','BqnG_Ei35JE','oaY9sMGtWmg','U1vGosMScjM',
                        'eu0KsZ_MVBc','zL3wWykAKfs','IJrKlSkxRHA','sn3cHUtNZKo','mO
KMoL8_jaQ','THpt6ugy_8E','LdyabrdFMC8','GNjStWG2vLU','Cjwoit91SxU',
                        'PHqqDu7fcVI','imrDkegrY1o','6XDwlQZKaK0','r7qovpFAGrQ','Re
cY5iZn6B0','hTGJfRPLe08','HLUX0y4EptA','lOfZLb33uCg','X9uk9IcoQ0w',
                        '8AHCfZTRGiI','HyHNuVaZJ-k','SYnVYJDxu2Q','NvS351QKFV4','fi
BLgEx6svA','SDTZ7iX4vTQ','FUXX55WqYZs']
def songs_to_data_ex5(ls_ids):

    ls_results = []

    for youtube_id in ls_ids:
        request = youtube.videos().list( #create the request for statistic and gene
ral information of an id
            part = "statistics, snippet",
            id = youtube_id)
        response = request.execute() #send the request and save the result in a var

        views = int(response['items'][0]['statistics']['viewCount']) #collect desir
ed data (name and views) from their respective location in the json file
        title = response['items'][0]['snippet']['title']
        ls_results.append([views, title])

    return sorted(ls_results, reverse=True) #sort the results by most views

result = songs_to_data_ex5(ls_recommended_songs)
y, x = zip(*result) #zip into two seperate lists for graphing


def halfwaysong(ls_in):
    '''
    finds the songs that divides the graph in left and right most evenly
    '''
    half_total_views = sum(y)/2 #what is halfway
    track_views = 0
    return_title = ""
    for views, title in ls_in:
        if (abs(half_total_views - track_views) > abs(half_total_views - (track_vie
ws+views))): #if the new itteration is closer to half than the previous
            track_views += views
            return_views = views
            return_title = title
        else:
            print("\n\nthe song that closest splits the graph 50/50 left and right
is : {}".format(return_title))
            songlistindex = ls_in.index([return_views, return_title])
            print("the nr. {} most popular song has {} views".format(songlistindex,
return_views) )
            return
    return

#plot the data
figex5 = go.Figure(data=go.Scatter(x=x, y=y, mode = 'lines'))
figex5.update_layout(
```



```
the song that closest splits the graph 50/50 left and right is : John Newman - L
ove Me Again
the nr. 2 most popular song has 724685542 views
```

Assignment 5 explanation

We would have liked to use the youtube API to gather recommended songs but since v3, youtube no longer allows users of the API to grab a list of recommended songs from another song. We believe this is because they want nobody to reverse their recommendation algorithm. Instead we decided to click on recommendations and their recommendations until we had a list of 50 IDs put them in a list and used that data instead.

we think the data is not super representative of youtube as a whole as more popular songs (>5m views) are recommended way more. The songs were selected through one of our personal accounts so our previous behaviour has also influenced the results.

This graph has the long tail attribute, the right side trails a long distrance towards 0. From the half way point we can see that the middle is pushed very far to the left, meaning that there is strong variation in music. A very large portion of the songs makes up for half of the interest, created by the many niches out there on youtube. And that is with a limited data set, where the least viewed song is not even below a million.

Assignment 6 - Conclusions

1) which effects and models explain best the data on music preferences

The data we collect in assignment 3 provides clear examples of power(-like) laws (we think the distribution might be an exponential one instead, for the YouTube data) for the distribution of views over music videos. We also think that we have found several interesting patterns in assignment 2, where we find that songs will always move toward some sort of predetermined like ratio, and that there are several "cost" functions that we can use to model music preferences.

We do not think that cascading effects can be a clear explanation for the trends in the data. We only have anecdotal evidence for some songs, and this is not enough to explain music preferences.

Even with the limited data assignment 5's graph really shows the long tail property.

2) which data we need to collect if we want to investigate cascading, network and rich-get-richer effects in music preferences?

We think that the data from YouTube videos provided us with much more interesting conclusions than the data from Spotify. A quantifiable number of likes, views, and recommended / related videos is much better for anaylsis than the arbitrary "popularity" number that Spotify uses. The most interesting data was obtained by comparing the number of views, and the like ratio of songs over time.

Thus, to investigate the effects mentioned above, one would at the very least need a metric like the number of views, and the like ratio (or a number that can be used to calculate the like ratio), and timestamps of the measurements. One could do analysis without the like ratio, but it is impossible to model any network effects without knowing the like ratio, as these were integral for our analysis in assignment 2.

It would be very interesting to investigate the network information diffusion in exercise 4 but this would require a large amount of additional data: people's individual music listening habbits and social structures next to their listening behaviour over a long period of time.