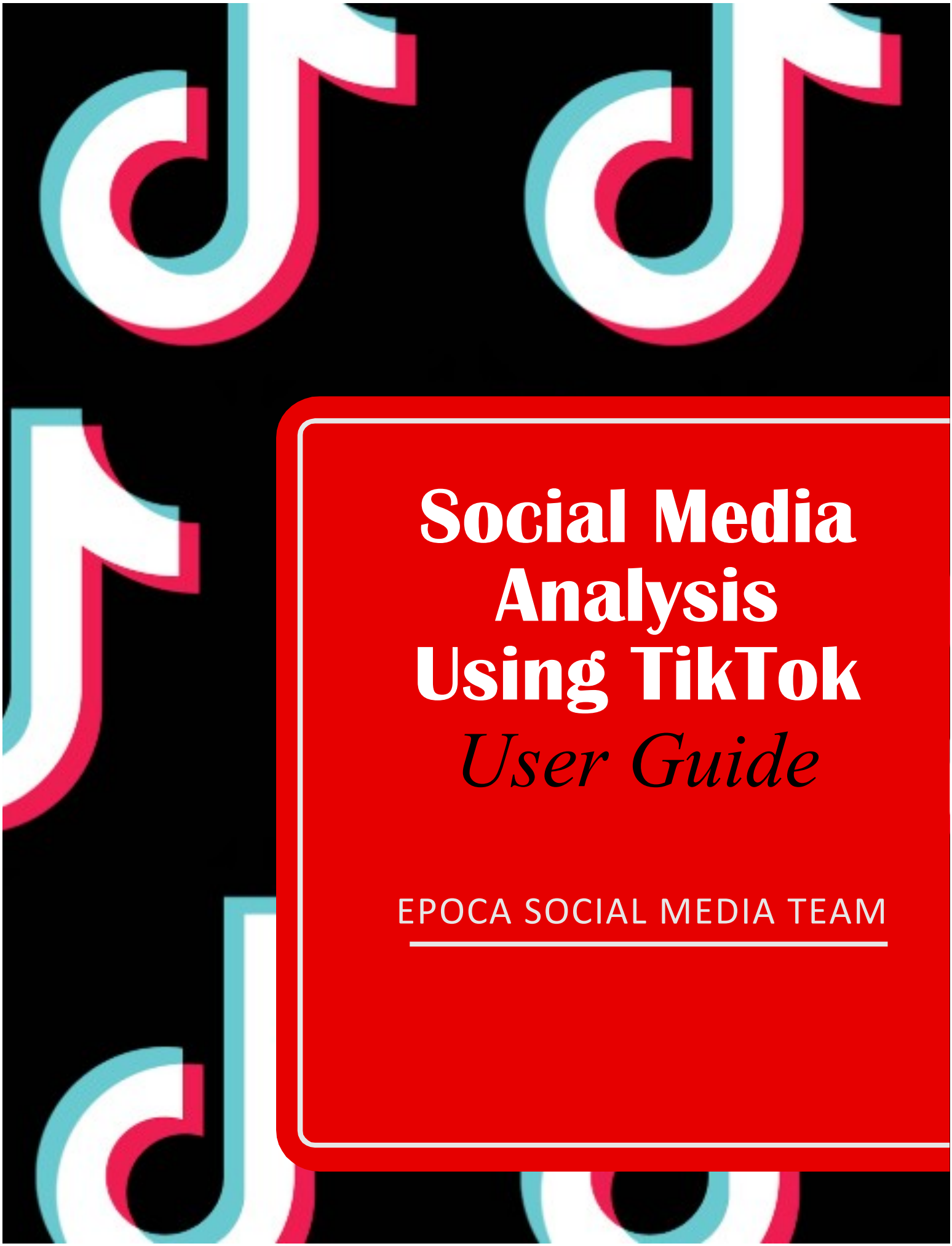


Social Media Analysis Using TikTok

User Guide

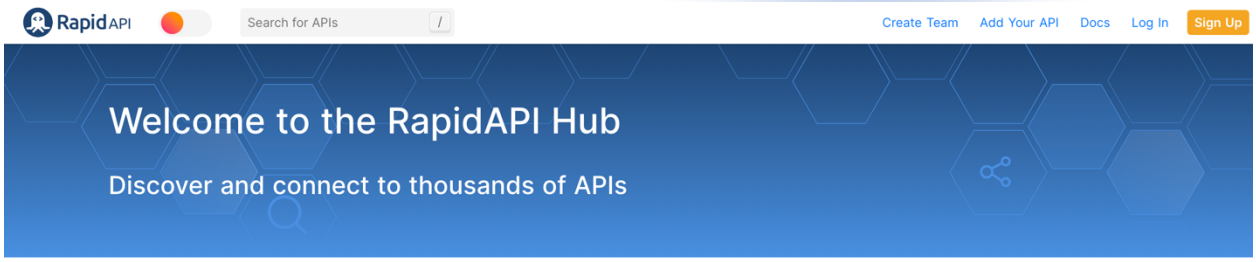
EPOCA SOCIAL MEDIA TEAM

Table of Contents

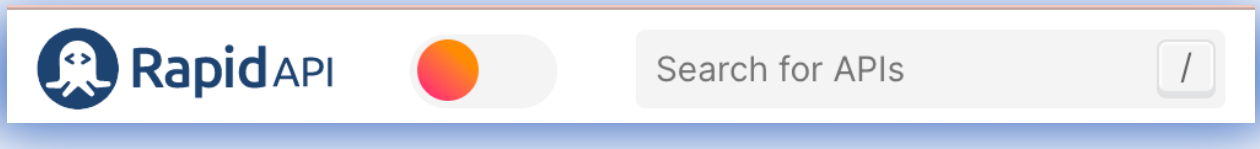
Pulling Data from TikTok API

1. Log into your preferred browser.

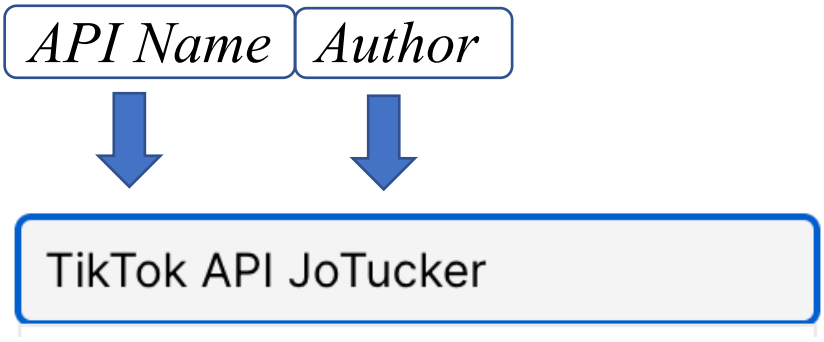
2. Navigate to RapidApi.com/hub
   Here you will be directed to a homepage where you can sign up.



3. Navigate to the top left corner of the webpage to the search engine.

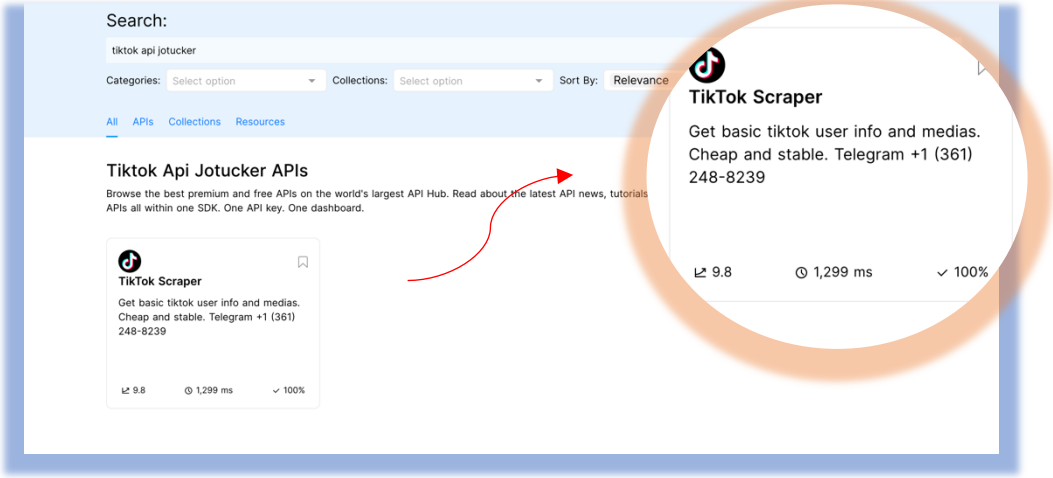


4. Search "TikTok API JoTucker".

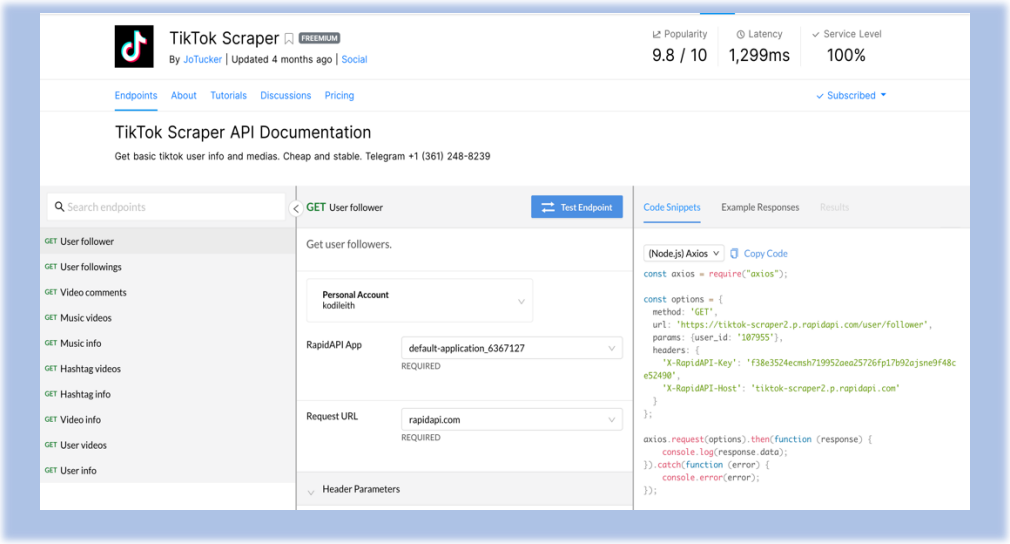


Pulling Data from TikTok API cont.

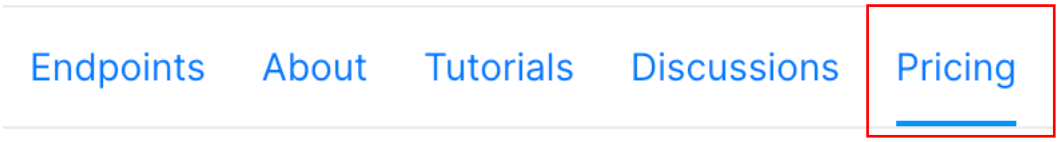
The result page should resemble the following.

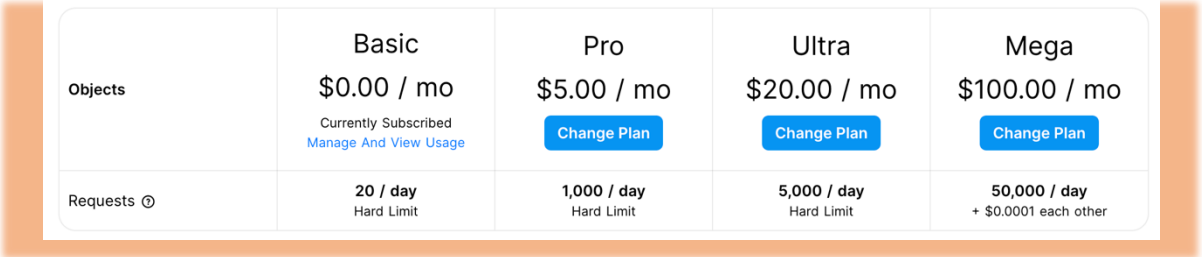


5. Double click to enter the API's Playground.



6. Navigate to the pricing subheading.





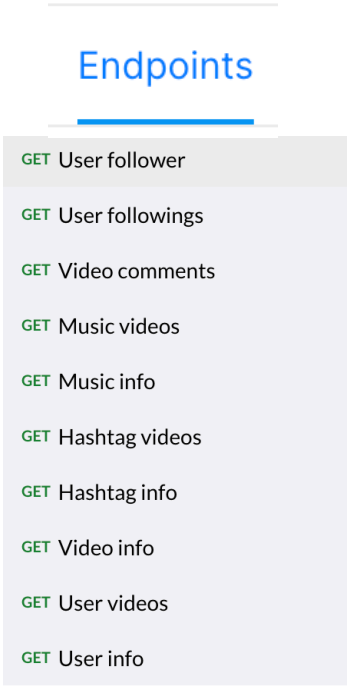
	Basic	Pro	Ultra	Mega
Objects	$0.00 / mo Currently Subscribed Manage And View Usage	$5.00 / mo Change Plan	$20.00 / mo Change Plan	$100.00 / mo Change Plan
Requests	20 / day Hard Limit	1,000 / day Hard Limit	5,000 / day Hard Limit	50,000 / day + $0.0001 each other

TikTok API offers different price tiers. Your chosen tier determines how many requests or pulls of data you can do in one day. {More on this later}

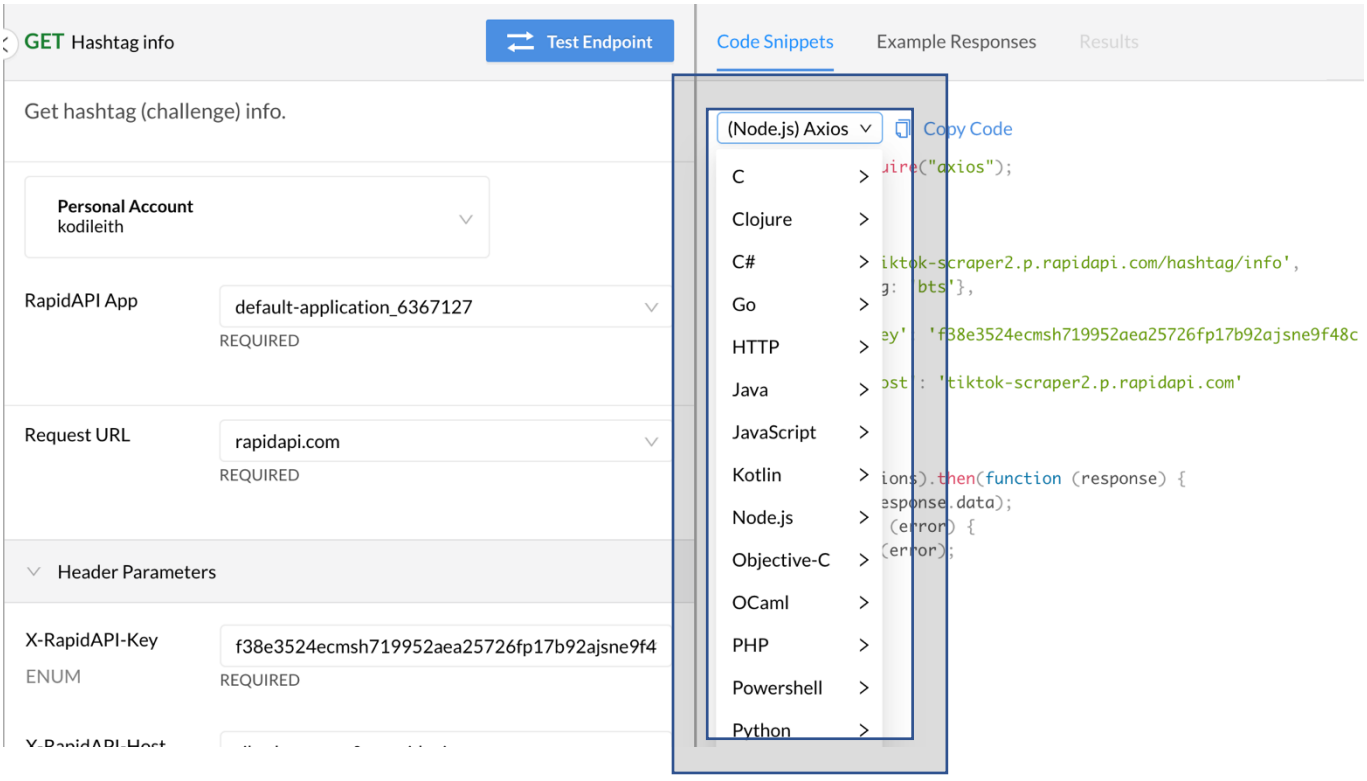
Pulling Data from TikTok API cont. 2

The endpoints subheading allows you to choose an endpoint to request from.

The available endpoints are shown here.



The endpoints give you options to retrieve code in your preferred language



Pulling Data from TikTok API cont. 3

This the API's default code given in python script to pull hashtag info from the API using the request package.

On our next page we will explain how we modified this code and created a loop.

(Python) Requests Copy Code

```
import requests

url = "https://tiktok-scraper2.p.rapidapi.com/hashtag/info"

querystring = {"hashtag":"bts"}

headers = {
    "X-RapidAPI-Key": "f38e3524ecmsh719952aea25726fp17b92ajsne9f48c
e52490",
    "X-RapidAPI-Host": "tiktok-scraper2.p.rapidapi.com"
}

response = requests.request("GET", url, headers=headers, params=que
rystring)

print(response.text)
```

Pulling Videos (Code)

We will start by importing our packages.

Requests is the method used to retrieve data from the TikTok API.

Requests package returns are given to us as a JSON string type and json.loads from the JSON package will be used to convert our JSON files to a python dictionary.

Math is a standard python package for computing basic math.

NumPy and Pandas will be used to manipulate the data and store it in data frames.

```
import requests
import json
import math

import numpy as np
import pandas as pd
```

Store your chosen list of hashtags into a variable.

```
# get hashtag ids from a list of hashtags - hashtag id is used to retrive videos for the next step

hashtag_names = ['kitchenutensil', 'kitchengoods','cookwares'] # You can customize your own hashtag list
```

```
get_hashid_url = "https://tiktok-scraper2.p.rapidapi.com/hashtag/info"
headers = {
  "X-RapidAPI-Key": "f823b922edmshcab34c928964e2dp1b4ce1jsnc0b4e45b7e3b",
  "X-RapidAPI-Host": "tiktok-scraper2.p.rapidapi.com"
}
```

This block of code is copied from the API's playground. These will be used as parameters in our request code.

Stores the hashtag information from the loop below into a list

```
hashtag_info_l = []
1 for name in hashtag_names:
  2 querystring = {"hashtag":name}
  3 hashid_response = requests.request("GET", get_hashid_url, headers=headers, params=querystring)

  4 hashtag_json = json.loads(hashid_response.text)
  5 hashid = hashtag_json['challengeInfo']['challenge']['id']
  6 video_count = hashtag_json['challengeInfo']['stats']['videoCount']

  7 print('hashtag: '+ name + "\tid: " + hashid + '\tvideos count: ' + str(video_count))
  8 hashtag_info_l.append((name,hashid,video_count))
```

The loop explained by line: **many of our future loops will replicate this format. This loop is explained in detail.**

1. Loops through every chosen hashtag in list "hashtag_names".
2. Saves a "querystring" variable which is a dictionary with the key as "hashtag" and the current hashtag being used in the loop or local variable as the value. This will be used as a parameter in our request code in the next line.
3. This line uses the requests package to pull data from the API and saves the JSON string response into a variable "hashid_response".
4. The hashid_response is returned as a JSON string. The package json.loads allow us to parse a JSON string and convert it to a python dictionary allowing it to be easier to read and manipulate.
5. From this converted dictionary we select the needed keys and their value. "Challenge Info", "challenge", and "id" are stored in "hashid".
6. From this converted dictionary we also select the keys pertaining to video count and their value. "Challenge Info", "Stats", and "videoCount" are store in the variable "video_count".
7. The hashtags name, Id and count of videos are printed in the console.
8. The hashtag name (local variable), "hashid" and "video_count" variables are then appended to our "hashtag_info_1" list which will be used in our next step.

Pulling Videos (Code) cont. 2

```
# a method to get videos from one hashtag

1 def getVideos(name, hashid, video_count):

    get_videos_url = "https://tiktok-scraper2.p.rapidapi.com/hashtag/videos"

    headers = {
        "X-RapidAPI-Key": "f823b922edmshcab34c928964e2dp1b4ce1jsnc0b4e45b7e3b",
        "X-RapidAPI-Host": "tiktok-scraper2.p.rapidapi.com"
    }

    # use video_count to identify pages
    # There might be a bug because the api is not able to retrive all the videos from each hashtag
2   pages = math.ceil(video_count / 30)

    # create lists to store videos' information
3   video_id_l = []
    video_desc_l = []
    video_likes_count_l = []
    video_comments_count_l = []
    video_timestamp_l = []
    video_url_l = []
    video_hashtags_l = []
```

```
# page loop
4 for i in range(pages):
    querystring = {"hashtag_id":hashid,"cursor":i*30}
    response_videos = requests.request("GET", get_videos_url, headers=headers, params=querystring)
    json_videos = json.loads(response_videos.text)

    try:
        video_items = json_videos['itemList']

        # on each page, get information of each video and save it to the lists created above
        for video in video_items:
            hashtags_l = []

            for hashtag_item in video['challenges']:
                hashtags_l.append(hashtag_item['title'])

            # get information of each video (id, description, time, url,etc)
            video_id = video['id']
            video_desc = video['desc']
            video_likes_count = video['stats']['diggCount']
            video_comments_count = video['stats']['commentCount']
            video_timestamp = video['createTime']
            video_author = video['author']['uniqueId']
            video_url = 'https://www.tiktok.com/@'+ video_author +'/video/'+ video_id # create video url using author name and video id
            video_hashtags = ' '.join(hashtags_l)
```

Once we have gotten a hashtag's information, we can now to pull videos and this function we created will do that.

1. This block of code is copied from the API's integration to be used in our request call.
2. This line identifies the pages in which the videos are listed on TikTok's platform.
3. This block of code creates list for the videos information by category for pulled data to be store later in the loop.
4. This loop parses through the pages and collects video information and saves it to variable category that mirrors our list categories to be later appended.

Pulling Videos (Code) cont. 3

```
1 # add video information to each list
  video_id_l.append(video_id)
  video_desc_l.append(video_desc)
  video_likes_count_l.append(video_likes_count)
  video_comments_count_l.append(video_comments_count)
  video_timestamp_l.append(video_timestamp)
  video_url_l.append(video_url)
  video_hashtags_l.append(video_hashtags)

    2 except:
          pass

      if json_videos == {}:
          break

      if json_videos['hasMore'] == False:
          break

  3 # save all videos information to a dictionary
    video_dic = {'video_id':video_id_l,
                 'video_desc':video_desc_l,
                 'likes_count': video_likes_count_l,
                 'comments_count': video_comments_count_l,
                 'video_timestamp':video_timestamp_l,
                 'video_url':video_url_l,
                 'video_hashtags':video_hashtags_l}

    # create a data frame for videos
    video_df = pd.DataFrame(video_dic)

    return video_df
```

We now need to add this video information to our list created earlier.

1. Append the collected video information in our variables to the group of lists created.
2. We don't want to save any videos that are missing needed information, so we include an except clause for this issue.
3. We now create a dictionary with all the video information that we have just pulled
4. This dictionary is then a converted to a Pandas data frame to be used in further analysis.

Pulling Videos (Code) cont. 4

```
# Some videos may be under more than one hashtag
# Create a method to delate the videos included in the previous hashtags

1 def deleteDuplicates(df1, df2):

    # create a list of the video ids in the previous data frame
    id1_l = list(df1['video_id'])

    # create a list of duplicated indexes
    del_index_l = []

    for i,id2 in enumerate(df2['video_id']):
        if str(id2) in id1_l:
            del_index_l.append(i)

    # drop duplicated rows
    df2 = df2.drop(index=del_index_l)

    return df2
```

```
# get all videos from hashtag list


2 for i,(name,hashid,video_count) in enumerate(hashtag_info_l):

    if i == 0:
        video_df = getVideos(name,hashid,video_count)
        # change long numbers to string format so the value will not be changed in the excel files
        video_df = video_df.astype({"video_id": str,'video_timestamp':str})
        video_df.to_excel("data/videos/videos_{}.xlsx".format(name), sheet_name='videos', index=False)

    else:
        df_sub = getVideos(name,hashid,video_count)
        df_sub = df_sub.astype({"video_id": str,'video_timestamp':str})
        df_sub = deleteDuplicates(video_df, df_sub)

        # save videos dataframe seprately for getting comments
        df_sub.to_excel("data/videos/videos_{}.xlsx".format(name), sheet_name='videos', index=False)

        # concat dfs
        video_df = pd.concat([video_df,df_sub], ignore_index=True)

3 # save dataframe of all videos to excel
video_df.to_excel("data/videos_all.xlsx",sheet_name='videos', index=False)

video_df
```

We need to remove duplicates from our list so that we aren't accounting for a piece of data more than once.

1. This function eliminates all duplicates. It parses through the list of our primary key, videos ids, and deletes any doubles.
2. This loop was created to collect videos from our selected hashtag list, format them and save them to a data frame.
3. Our now cleaned of duplicates data frame is saved as an excel file to our selected directory.

Pulling Comments (Code)

We will start by importing our packages.

Requests is the method used to retrieve data from the TikTok API.

Requests package returns are given to us as a JSON string type and json.loads from the JSON package will be used to convert our JSON files to a python dictionary.

Math is a standard python package for computing basic math.

Time is python it is useful for many time conversions.

Re or regular expressions allows you to check for string matches or patterns.

Glob is used to return all file paths that match a specific pattern.

NumPy and Pandas will be used to manipulate the data and store it in data frames.

```
import requests
import json
import math
import time
import glob
import re

import numpy as np
import pandas as pd
```

We use Glob to import and sort our files containing our videos.

```
files = glob.glob(r'data/videos/*.xlsx')
files.sort()
files
```

Pulling Comments (Code) cont. 2

```
1 def getComments(videos_df):

    get_comments_url = "https://tiktok-scraper2.p.rapidapi.com/video/comments"
    headers = {
        "X-RapidAPI-Key": "f823b922edmshcab34c928964e2dp1b4ce1jsnc0b4e45b7e3b",
        "X-RapidAPI-Host": "tiktok-scraper2.p.rapidapi.com"
    }

    c_video_id_l = []
    comment_id_l = []
    comment_timestamp_l = []
    comment_text_l = []

2   for i,video_url in enumerate(videos_df['video_url']):

        pages = math.ceil(videos_df['comments_count'][i] / 50)

        # on each page, get information of each comment and save it to the lists created above
        for page in range(pages):
            querystring = {"video_url":video_url,"count":"50","cursor":page*50}
            response_comments = requests.request("GET", get_comments_url, headers=headers, params=querystring)
            time.sleep(1)
            json_comments = json.loads(response_comments.text)

            try:
                comment_items = json_comments['comments']
                # cursor = json_comments['cursor']
                 # get information of each comment (video id, comment id, time, text)
                 for comment in comment_items:
                      c_video_id = comment['aweme_id']
                      comment_id = comment['cid']
                      comment_timestamp = comment['create_time']
                      comment_text = comment['text']

                      # add comment information to each list
                      c_video_id_l.append(c_video_id)
                      comment_id_l.append(comment_id)
                      comment_timestamp_l.append(comment_timestamp)
                      comment_text_l.append(comment_text)

       3 except:
                 continue

            if json_comments == {}:
                 break

            if json_comments['has_more'] == 0:
                 break

4 # save all comments information into a dictionary
  comment_dic = {'c_video_id': c_video_id_l,
                 'comment_id': comment_id_l,
                 'comment_timestamp': comment_timestamp_l,
                 'comment_text': comment_text_l
                }
```

We created a function to pull comments based on a list of videos provided.

1. **We create a list to store our comment data once returned this includes the video it is commented withs id, the comments id, timestamp, and the actual text.**
2. **We will not accept any empty comment fields.**
3. **All comments are saved to a dictionary with the required categories.**

Pulling Comments (Code) cont. 3

```
1  # transfer to dataframe
   comment_df = pd.DataFrame(comment_dic)

2  # change long numbers to string format so the value will not be changed in the excel files
   comment_df = comment_df.astype({"c_video_id": str, 'comment_id': str, 'comment_timestamp':str})

   return comment_df
3 for file in files:

      # get hashtag name from the file name
      hashtag_name = re.findall('([a-z0-9]+)\.xlsx',file)[0]

      videos_df = pd.read_excel(file)

      comment_df = getComments(videos_df)

      # save comments table to local files
      comment_df.to_excel("data/comments/comments_{}.xlsx".format(hashtag_name), sheet_name='comments', index=False)
```

1. **Converts the dictionary with our comments to a Pandas data frame.**
2. **Converts our numbers to a string so they will not be altered when saved as an excel file.**
3. **This loop will find the hashtag name in our files from earlier and save our comments Pandas data frame into our local files as an excel file.**

Speech Recognition (Code)

We will start by importing our packages.

OS provides functions for interacting with the operating system.

Selenium package automates browsers.

BeautifulSoup is a standard python package for pulling data out of HTML and XML files.

Moviepy.editor is python module for video editing. (Cuts, concatenation, inserts)

Time is python it is useful for many time conversions.

Speech Recognition recognizes speech for transcribing audio.

Pandas will be used to manipulate the data and store it in data frames.

```
import os
# you may need to install selenium
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
import urllib.request
from bs4 import BeautifulSoup
import moviepy.editor as mp
import speech_recognition as sr
import time

import pandas as pd
```

1
```
df = pd.read_excel("data/videos_all.xlsx")
```

2
```
driver = webdriver.Chrome(r'/Users/Flame/Dropbox/Summer/QMB6930/chromedriver') # local path of chromedriver file
# A new chrome window will pop out
```

3
```
# Set directory
os.chdir('/Users/Flame/Dropbox/Summer/QMB6930') # your folder path
```

4
```
# Create a temp directory
if not os.path.exists("./temp"):
    os.makedirs("./temp")
```

1. Import our list of videos.
2. The local path of our Chromedriver. Chromedriver will open a window in which the videos will run in a loop as the audio transcriber works.
3. Sets the directory must be the same folder in which your downloaded chrome driver will be.
4. Created a temporary directory in which the videos will be download so that we can run the audio.

Speech Recognition (Code) cont. 2

```
result_l = []

# this code may run for hours
i = 0
for url in df['video_url']:

    i += 1
    driver.get(url)

    # give chrome some time to load the page
    time.sleep(2)

    # 1. get page_source
    # 2. Parse HTML
    # 3. Find the node for the video
    video_url = BeautifulSoup(driver.page_source, 'html.parser').find("video")

    try:
        # Download video and save as "./temp/temp.mp4"
        urllib.request.urlretrieve(video_url["src"], './temp/temp.mp4')

        # Rip audio from the video
        clip = mp.VideoFileClip('./temp/temp.mp4')
        clip.audio.write_audiofile('./temp/temp.wav')

        # Run Speech Recognition
        r = sr.Recognizer()
        audio_file = sr.AudioFile('./temp/temp.wav')

        with audio_file as source:
            r.adjust_for_ambient_noise(source)
            audio = r.record(source)

        result = r.recognize_google(audio)

    except:
        result = ''

    result_l.append(result)
    print('video {} is done.'.format(str(i)))
```

1. Create a list for our transcript result to be saved.
2. This loop uses chrome driver as a separate tab which will loop through the videos and transcribe the videos audio into a text format.
3. Does not accept empty audio transcripts.
4. The resulting audio transcript is appended to our result list. Then a message is printed to the console letting you know the current video has been transcribed.

Speech Recognition (Code) cont. 3

```
1 # close the driver
  driver.quit()
```

```
2 # make sure the outcome is True
  len(result_l) == len(df['video_url'])
```

```
3 # add the transcripts to a new column
  df['video_transcript'] = result_l

4 # change video id and timestamp to str type
  df = df.astype({"video_id": str, 'video_timestamp': str})
```

```
5 # save the new dataframe as an excel file (replace the original one)
  df.to_excel("videos_all.xlsx",sheet_name='videos', index=False)
```

1. Closes the driver so that it is no longer running in the background of your computer.
2. Checks to see if the length of the data frame with the transcript is the same as our original data frame to make sure each video was converted.
3. The transcript text will be appended as a new column.
4. The video id and timestamps need to be converted to strings for us to later format them.
5. This new data frame with transcripts attached is converted and saved as an excel file.

NLP, Bigrams and Colors Analysis (Code)

Our next section is commented heavily for guidance.

```
!pip install webcolors
```

```
import glob
import numpy as np
import pandas as pd
import nltk
import re
import cld2 # detect language
import webcolors # the color dictionary
import collections # for counting bigrams from the list

from nltk.corpus import stopwords # stop word list
from nltk.stem import WordNetLemmatizer # Lemmentize words
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer # Tokenize words by space

from nltk.util import ngrams # for bigrams
from nltk.collocations import * # for bigrams
from nltk import FreqDist # for getting the most frequent words

import time
import datetime as dt
from datetime import datetime
from dateutil.relativedelta import relativedelta # for compute the date before 6 months

import matplotlib.pyplot as plt
from wordcloud import WordCloud # for wordcloud
```

```
nltk.download('stopwords') # for stopwords list
nltk.download('punkt') # for word tokenize
nltk.download('genesis') # for bigrams
nltk.download('words')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
# This is the time of the whole dataset (2 years)
start_date = '2020-6-1'# starts at 2020-6-1 00:00(included)
end_date = '2022-6-1' # ends at 2022-6-1 0:00(not included)

# This is the number of months we want to detect trends
detected_months = 6
```

```
# define a method to transfer date-time to timestamp
def toTimestamp(d):
    result  = time.mktime(dt.datetime.strptime(d,'%Y-%m-%d').timetuple())
    return int(result)
```

NLP, Bigrams and Colors Analysis (Code) cont. 2

```
# list of stopwords
stopWords = list(stopwords.words('english'))

# Extend the list with your own custom stopwords
my_stopwords = [
    '699nt','absolutely','account','actually','add','ago','agree','aku','almost','already','also','always',
    'amazing','amazon','another','anyone','anything','apa','apartnment','around','ask','available','away',
    'awesome','back','bad','baru','bass','beautiful','believe','best','better','bit','boom','bought','brand',
    'break','bring','buy','call','came','can','cannot','cant','care','cause','changer','check','chef','clean',
    'click','code','come','comes','coming','comment','comments','cooking','cool','could','course','cute','dah',
    'damn','dari','day','days','definitely','die','different','discount','dollar','done','dont','dream','dude',
    'easier','easily','easy','eat','either','eneryone','enjoy','enough','even','ever','every','everyone',
    'everything','exact','exactly','excited','far','favorite','feel','finally','find','fine','first','follow',
    'food','found','free','friend','full','fun','funny','future','gadget','game','get','getting','girl','give',
    'glad','god','going','gonna','good','gorgeous','gosh','got','grab','grandma','great','guess','guna','haha','happy',
    'hate','hear','hell','hello','help','hey','high','highly','home','honestly','hont','hope','house','human',
    'husband','idea','item','job','kan','keep','kind','kitchen','kitchenware','knew','know','ladies','lady','lama',
    'last','learned','leave','let','life','like','link','list','literally','little','live','lol','look','looking',
    'looks','lot','love','made','make','making','man','mana','many','mau','may','maybe','mean','might','mine',
    'miss','mom','money','much','must','nak','name','nan','need','needs','never','new','next','nextback','nice',
    'not','nothing','oh','okay','omg','one','order','ordered','para','part','people','perfect','person','photo',
    'play','please','plus','point','post','pretty','price','probably','process','product','profile','purchase',
    'put','quality','question','ready','real','really','recommend','recommendation','remember','reply','right',
    'said','sale','save','saw','say','saying','search','second','see','seeing','seen','sell','send','sense',
    'series','share','shipping','shop','shopping','show','si','similar','sis','sold','someone','something','soon',
    'sorry','start','stay','still','stop','stuff','super','sure','take','tell','thank','thanks','thats','thing',
    'think','thinking','tho','though','thought','time','today','toko','top','tree','true','try','understand','use',
    'used','usually','video','videos','wait','waiting','want','watch','watching','way','week','weird','well',
    'went','wish','without','wonder','work','works','world','worth','would','wow','wrong','yang','yea','yeah',
    'year','yes','yet'
]
stopWords.extend(my_stopwords)
```

```
# define a method to detect English text (used to remove non-English text)

def detectEnglish(str):
    try:
        isReliable, textBytesFound, details = cld2.detect(str)

        # including 'Unknown' - some English words or numbers might be detected as 'Unknown'
        if details[0][0] == 'ENGLISH' or details[0][0] == 'Unknown':
            return True
        else:
            return False
    except:
        return True
```

NLP, Bigrams and Colors Analysis (Code) cont. 3

```
# define a method using nlp to clean the video dataframe

def cleanVideosDF(df,start_date,end_date):

    # Only keep videos from the start date to the end date (2 years)
    df = df[(df['video_timestamp'] >= toTimestamp(start_date)) & (df['video_timestamp'] < toTimestamp(end_date))]

    # in the video captions, remove all the hashtags, mentions, and web links
    df['video_desc'] = df['video_desc'].apply(lambda x: re.sub(r'#\S*', '', x))
    df['video_desc'] = df['video_desc'].apply(lambda x: re.sub(r'@\S*', '', x))
    df['video_desc'] = df['video_desc'].apply(lambda x: re.sub(r'https:\S*', '', x))

    # Only keep video captions which are in English
    df = df[df['video_desc'].apply(detectEnglish) == True]

    # join the video caption and transcript; transfer all the words to lowercase
    df['video_text'] = (df['video_desc'].map(str) + ' '+ df['video_transcript'].map(str)).astype(str).str.lower()

    # Lemmatize words
    wordnet_lem = WordNetLemmatizer()
    df['text_string'] = df['video_text'].apply(wordnet_lem.lemmatize)

    # \w+ matches Unicode word characters with one or more occurrences
    # this includes most characters that can be part of a word in any language, as well as numbers and the underscore.
    regexp = RegexpTokenizer('\w+')

    # Tokenize - split scentenses into words
    df['text_token'] = df['text_string'].apply(regexp.tokenize)

    # Remove stopwords
    df['text_token'] = df['text_token'].apply(lambda x: [w for w in x if w not in stopWords])
```

```
# Only keep English words and numbers
en_words = set(nltk.corpus.words.words())
df['text_token'] = df['text_token'].apply(lambda x: [w for w in x if w.isnumeric() or w in en_words])
df['text_token'] = df['text_token'].apply(lambda x: [w for w in x if w.isnumeric() or w in en_words])

# join the selected words
df['text_string'] = df['text_token'].apply(lambda x: ' '.join([item for item in x if len(item)>2]))

# drop blank rows after text cleaning
df = df.drop(df[df['text_string'] == ''].index)

return df
```

NLP, Bigrams and Colors Analysis (Code) cont. 4

```
# Import videos dataframe
videos_df = pd.read_excel("data/videos_all.xlsx")
videos_df_clean = cleanVideosDF(videos_df,start_date,end_date)
```

```
videos_df_clean[['video_text', 'text_token', 'text_string']]
```

```
# define a method using nlp to clean the comment dataframe
def cleanCommentsDF(df,start_date,end_date):

    df = df[(df['comment_timestamp'] >= toTimestamp(start_date)) & (df['comment_timestamp'] < toTimestamp(end_date))]

    # Only keep comment texts which are in English
    df = df[df['comment_text'].apply(detectEnglish) == True]

    # transfer all the words to lowercase
    df['comment_text'] = df['comment_text'].astype(str).str.lower()

    # in the video caption, remove all the hashtags, mentions, and web links
    df['comment_text'] = df['comment_text'].apply(lambda x: re.sub(r'#\S*', '', x))
    df['comment_text'] = df['comment_text'].apply(lambda x: re.sub(r'@\S*', '', x))
    df['comment_text'] = df['comment_text'].apply(lambda x: re.sub(r'https:\S*', '', x))

    # Lemmatize words
    wordnet_lem = WordNetLemmatizer()
    df['text_string'] = df['comment_text'].apply(wordnet_lem.lemmatize)

    # \w+ matches Unicode word characters with one or more occurrences
    # this includes most characters that can be part of a word in any language, as well as numbers and the underscore.
    regexp = RegexpTokenizer('\w+')

    # Tokenize - split scentenses into words
    df['text_token']=df['text_string'].apply(regexp.tokenize)

    # Remove stopwords
    df['text_token'] = df['text_token'].apply(lambda x: [w for w in x if w not in stopWords])
```

```
# Only keep English words and numbers
en_words = set(nltk.corpus.words.words())
df['text_token'] = df['text_token'].apply(lambda x: [w for w in x if w.isnumeric() or w in en_words])
df['text_token'] = df['text_token'].apply(lambda x: [w for w in x if w.isnumeric() or w in en_words])

# join the selected words
df['text_string'] = df['text_token'].apply(lambda x: ' '.join([item for item in x if len(item)>2]))

# drop blank rows after text cleaning
df = df.drop(df[df['text_string'] == ''].index)

return df
```

NLP, Bigrams and Colors Analysis (Code) cont. 5

```
# Import file
files = glob.glob(r'data/comments/*.xlsx')
files.sort()
files
```

```
for i, file in enumerate(files):
    df_sub = pd.read_excel(file)
    df_sub_clean = cleanCommentsDF(df_sub,start_date, end_date)

    if i == 0:
        comments_df_clean = df_sub_clean
    else:
        comments_df_clean = pd.concat([comments_df_clean, df_sub_clean], ignore_index=True)
```

```
comments_df_clean[['comment_text', 'text_token', 'text_string']]
```

```
# save comments frame to trace back the video links for trend analysis

# transfer long numbers to string type
comments_df_clean = comments_df_clean.astype({'c_video_id': str, 'comment_id':str})

# transfer timestamp to yeat-month format
comments_df_clean['month'] = comments_df_clean['comment_timestamp'].apply(lambda x: datetime.fromtimestamp(x).date().isoformat())
comments_df_clean['month'] = pd.to_datetime(comments_df_clean['month']).dt.strftime('%Y-%m')

comments_df_clean.to_excel('data/comments_all_clean.xlsx', index=False)
```

NLP, Bigrams and Colors Analysis (Code) cont. 6

```
# Only keep the time and text parts of the dataframes

videos_df_clean2 = videos_df_clean[['video_timestamp', 'text_token', 'text_string']]
videos_df_clean2 = videos_df_clean2.rename(columns={'video_timestamp':'timestamp'})

comments_df_clean2 = comments_df_clean[['comment_timestamp', 'text_token', 'text_string']]
comments_df_clean2 = comments_df_clean2.rename(columns={'comment_timestamp':'timestamp'})

# concat videos and comments texts
df_clean = pd.concat([videos_df_clean2,comments_df_clean2], ignore_index=True)
```

```
# define a method to get the start date of detecting trends (6 months before the end date of the whole dataset)

def getDetectedDateStarts(end_date,detected_months):
    date_obj = datetime.strptime(end_date, '%Y-%m-%d')
    detected_date_starts = date_obj + relativedelta(months=-detected_months)
    detected_date_starts = detected_date_starts.strftime('%Y-%m-%d')

    return detected_date_starts
```

```
# get the dataframe for the specific period (the recent 6 months)

def getStatDf(df, end_date, detected_months):
    start_date = getDetectedDateStarts(end_date,detected_months)
    df = df[(df['timestamp'] >= toTimestamp(start_date)) & (df['timestamp'] < toTimestamp(end_date))]
    return df
```

```
# create a dataframe used to count the most frequent bigrams in the recent 6 months
df_stat = getStatDf(df_clean, end_date, detected_months)
df_stat
```

```
# join all words
all_words = ' '.join([word for word in df_stat['text_string'] ] )
```

NLP, Bigrams and Colors Analysis (Code) cont. 7

```
# Define a method to get most frequent bigrams
def getBigrams(df, no_of_bigrams):
    all_bigrams_l = []
    for strings in df['text_string']:
        # split the words
        tokens = strings.split(' ')
        if len(tokens) >= 2:
            bigrams = list(ngrams(tokens,2))
            for bigram in bigrams:
                all_bigrams_l.append(bigram)

    # join the bigram words
    for i, (a,b) in enumerate(all_bigrams_l):
        all_bigrams_l[i] = a + ' ' + b

    # get the most ferquent n bigrams
    bigram_l = collections.Counter(all_bigrams_l).most_common(no_of_bigrams)

    for i, (a,b) in enumerate(bigram_l):
        bigram_l[i] = a

    return bigram_l
```

```
# set a number of the most frequent bigrams
no_of_bigrams =20

# get bigram list
bigram_l = getBigrams(df_stat, no_of_bigrams)
bigram_l
```

```
# define a method to return a dataframe that can be used for trend analysis

def getBigramDF(df, bigram_l):
    v_dates_l = []
    v_bigrams_l = []

    for i,string in enumerate(df['text_string']):
        for w in bigram_l:
            if w in re.findall(w, string):
                v_dates_l.append(df['timestamp'][i])
                v_bigrams_l.append(w)

    df_bigram = pd.DataFrame(list(zip(v_dates_l, v_bigrams_l)),columns = ['month', 'bigram'])

    df_bigram['month'] = df_bigram['month'].apply(lambda x: datetime.fromtimestamp(x).date().isoformat())
    df_bigram['month'] = pd.to_datetime(df_bigram['month']).dt.strftime('%Y-%m')

    return df_bigram
```

```
# get the time and bigram dataframe and save it for trend analysis
df_bigram = getBigramDF(df_clean, bigram_l)
df_bigram.to_excel('data/bigrams_for_trend_analysis.xlsx', index=False) # for trend detection
```

Product Bigram Trend Analysis (Code)

Our next section is commented heavily for guidance.

```
import pandas as pd
import numpy as np
import statistics as stat
import matplotlib.pyplot as plt
from scipy.signal import find_peaks # for detecting peaks
```

```
# Open file
df = pd.read_excel("data/bigrams_for_trend_analysis.xlsx")

# group data frame by month and bigrams
df_grouped = df.groupby(['month','bigram'])['bigram'].count().unstack()

# replace the null values with 0
df_grouped.replace(np.nan, '0',inplace=True)
df_grouped = df_grouped.astype(int)
df_grouped
```

OUTPUT EXAMPLE:

bigram	air fryer	baking soda	boiling water	butcher block	carbon steel	cast iron	cheese grater	cinnamon toast	classic vintage	corner cabinet	...	rice dispenser	salt pepper	scrub daddy	spice cabinet	spice rack	stainless steel
month																	
2020-06	2	0	0	0	0	0	0	0	0	0	...	0	2	0	1	7	5
2020-07	3	2	2	0	0	2	1	0	0	1	...	0	4	0	1	12	3
2020-08	2	1	0	0	0	0	0	0	0	1	...	0	5	0	0	1	1
2020-09	46	0	1	0	0	11	2	0	0	5	...	0	0	0	0	2	4
2020-10	3	4	4	1	0	26	0	0	0	2	...	0	4	0	0	4	3
2020-11	35	2	2	2	1	6	0	1	0	1	...	0	2	0	5	0	3

Product Bigram Trend Analysis (Code) cont. 2

```
# This gives you all the bigrams in the dataframe
# you can only keep the bigrams you are interested in and paste them into the list in next box
df_grouped.columns
```

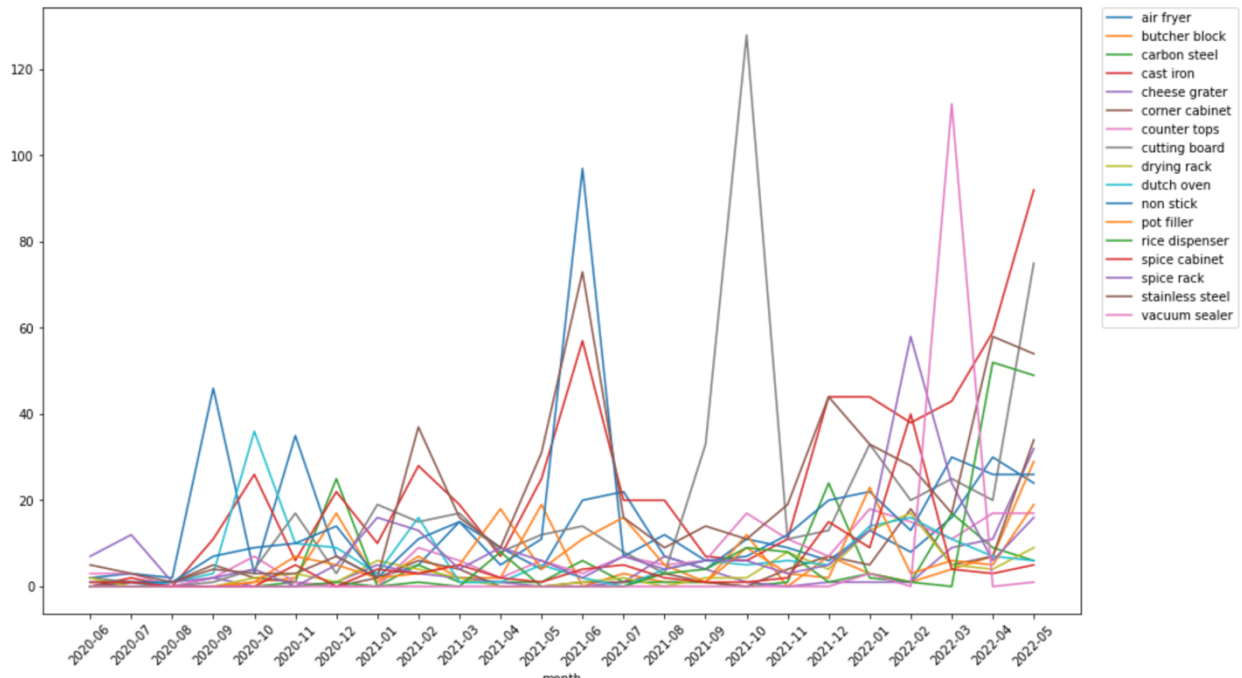
Output example:

```
Index(['air fryer', 'baking soda', 'boiling water', 'butcher block',
       'carbon steel', 'cast iron', 'cheese grater', 'cinnamon toast',
       'classic vintage', 'corner cabinet', 'counter tops', 'crunch seasoning',
       'crunch spice', 'cumin powder', 'custom wood', 'cutting board',
       'dining room', 'drying rack', 'dutch oven', 'empty space',
       'expiration date', 'garlic powder', 'hot water', 'ice cream',
       'meal prep', 'non stick', 'olive oil', 'paper towel', 'pot filler',
       'potato salad', 'rice dispenser', 'salt pepper', 'scrub daddy',
       'spice cabinet', 'spice rack', 'stainless steel', 'toast crunch',
       'useful tool', 'vacuum sealer', 'wood burning'],
      dtype='object', name='bigram')
```

```
# This is an option to customize your own list of bigrams
df_grouped = df_grouped[['air fryer', 'butcher block','carbon steel', 'cast iron', 'cheese grater',
                         'corner cabinet', 'counter tops', 'cutting board', 'drying rack', 'dutch oven',
                         'non stick', 'pot filler', 'rice dispenser', 'spice cabinet', 'spice rack',
                         'stainless steel', 'vacuum sealer'
                        ]]
```

TREND OVERVIEW FOR ALL BIGRAMS:

```
# an overview of the graph of all bigrams
# fig,ax = plt.subplots(figsize=(15,9))
df_grouped.plot(figsize=(15,9))
plt.xticks(np.arange(len(df_grouped.index)), df_grouped.index, rotation=45)
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
```



Product Bigram Trend Analysis (Code) cont. 3

TREND DETECTION: 3 TYPES

```
# This is the month priod of detecting trend
detect_month = 6
```

```
# define a method to get slopes of the line
def getSlope(list_of_index, array_of_data, order=1):
    result = np.polyfit(range(len(list_of_index)), array_of_data, order)
    slope = result[-2]
    return float(slope)
```

TREND TYPE 1: NEW TREND
-No data before 6 months
-There are some data during recent 6 months
-The slope for 2-year data is upward

```
def newTrendDetector(df,detect_month,bigram_name):
    boo_list = []
    for value in df[bigram_name]:
        if value == 0:
            boo_list.append(0)
        else:
            boo_list.append(1)
    if sum(boo_list[:len(boo_list)-detect_month]) == 0 and sum(boo_list[-detect_month:]) != 0 and getSlope(df.index,df[bigram_name]) > 0:
        return True
    else:
        return False
```

Product Bigram Trend Analysis (Code) cont. 4

TREND TYPE 2: PEAKED TREND
-There is a peak within the recent 6 months

```
def peakedTrendDetector(df,detect_month,bigram_name):

    # method to get the peak value
    # define the value of prominence: 50% percent of the max value in the recent 6 months,
    prominence = df[bigram_name][-detect_month:].max()*0.5
    peaks, _ = find_peaks(df[bigram_name][-detect_month:], prominence=prominence)

    # if there is a peak, return a list of peaked months
    if peaks.size > 0:
        peaks, _ = find_peaks(df[bigram_name], prominence=10)

        peaked_months_l = []
        for peak in peaks:
            peaked_month = df.index[peak]
            peaked_months_l.append(peaked_month)
        return peaked_months_l

    else:
        return False
```

TREND TYPE 3: STEP CHANGE
-There is no peak during the recent 6 months
-There are some data before the 6 months
-During the recent 6 months, there is at least one month whose 3-month standard deviation. Of moving average is greater than 2.
-The slope of the recent 6 months is positive

Bigram Trend Analysis (Code) cont. 5

```
def stepChengeDetector(df,detect_month,bigram_name):
    move_avg_l = []
    std_l = []
    boo_list = []

    # calculate moving average in 3 months
    for i,value in enumerate(df[bigram_name]):
        if i <= 1:
            avg = stat.mean(df[bigram_name][0:i+1])
        else:
            avg = stat.mean(df[bigram_name][i-2:i+1])
        move_avg_l.append(avg)

    # calculate standard deviation for the moving average in 3 months
    for i,value in enumerate(move_avg_l):
        if i == 0:
            std = 0
        else:
            if i == 1:
                std = stat.stdev(move_avg_l[0:2])
            else:
                std = stat.stdev(move_avg_l[i-2:i+1])
        std_l.append(std)

    for x in std_l:
        if x > 2:
            boo_list.append(1)
        else:
            boo_list.append(0)
```

```
# Make sure it is not a recent peaked trend
if peakedTrendDetector(df,detect_month,bigram_name) == False:

    if sum(df[bigram_name][:len(boo_list)-detect_month]) != 0 and sum(boo_list[len(boo_list)-detect_month:]) != 0 and getSlope(df.index[-detect_month:],
        return True

    else:
        return False
else:
    return False
```

Bigram Trend Analysis (Code) cont. 6

Continued.

```
# create lists to save the bigrams of each type of trend
new_trend_l = []
peaked_trend_l = []
step_change_l = []

# create a dictionary to save the bigrams' peaked month
bigram_and_peaked_months = {}

for bigram_name in df_grouped.columns:

    if newTrendDetector(df_grouped,detect_month,bigram_name) != False:
        new_trend_bigram = bigram_name
        new_trend_l.append(new_trend_bigram)

    if peakedTrendDetector(df_grouped,detect_month,bigram_name) != False:
        peaked_bigram = bigram_name
        peaked_trend_l.append(peaked_bigram)
        bigram_and_peaked_months[bigram_name] = peakedTrendDetector(df_grouped,detect_month,bigram_name)

    if stepChengeDetector(df_grouped,detect_month,bigram_name) != False:
        step_change_bigram = bigram_name
        step_change_l.append(step_change_bigram)
```

Examples of our three trend detector functions when ran on our data.

```
new_trend_l
```

```
['vacuum sealer']
```

```
peaked_trend_l
```

```
['drying rack',
 'dutch oven',
 'pot filler',
 'spice cabinet',
 'spice rack',
 'vacuum sealer']
```

```
step_change_l
```

```
['air fryer',
 'butcher block',
 'cast iron',
 'cheese grater',
 'corner cabinet',
 'cutting board',
 'non stick',
 'rice dispenser',
 'stainless steel']
```

Bigram Trend Analysis (Code) cont. 7

```
# Define a method to save bigrams trend graph to local folder
def saveTrendFig(df_grouped, trend_type, bigram_list):

    for bigram in bigram_list:
        df_single_bigram = df_grouped[[bigram]]

        df_single_bigram.plot(figsize=(15,9))
        plt.title(bigram,fontsize=20)
        plt.legend().set_visible(False)
        # plt.gcf().autofmt_xdate()
        plt.xticks(np.arange(len(df_grouped.index)), df_grouped.index, rotation=45)

        plt.savefig('data/graphs/bigrams/{} ({}).png'.format(trend_type,bigram), facecolor = 'white')
        plt.close()
```

```
# save graphs of new trend bigrams
saveTrendFig(df_grouped, 'new_trend', new_trend_l)

# save graphs of peaked trend bigrams
saveTrendFig(df_grouped, 'peaked_trend', peaked_trend_l)

# save graphs of new trend bigrams
saveTrendFig(df_grouped, 'step_change', step_change_l)
```

Get Video links of Peaked Bigrams:

```
# dictionary of peaked bigrams and their peaked months
bigram_and_peaked_months
```

```
{'drying rack': ['2022-02'],
 'dutch oven': ['2020-10', '2021-02', '2022-02'],
 'pot filler': ['2021-04', '2021-07', '2022-01'],
 'spice cabinet': ['2022-02'],
 'spice rack': ['2021-01', '2022-02'],
 'vacuum sealer': ['2022-03']}
```

```
# open the comments dataframe
comments_df = pd.read_excel('data/comments_all_clean.xlsx')
comments_df[:10]

# open the video dataframe to get the videos information
videos_df = pd.read_excel("data/videos_all_clean.xlsx")
videos_df.replace(np.nan, '',inplace=True)
videos_df
```

Bigram Trend Analysis (Code) cont. 8

```
def getVideoRefDF(videos_df, comments_df, bigram_months_dic):


    bigram_l = []
    month_l = []
    video_id_l = []

    # Step 1
    # check each comment, see if it contains the biagram
    # save the bigram name, month, and video id

    for i,comment_text in enumerate(comments_df['comment_text']):

        for bigram in bigram_months_dic:

            for month in bigram_months_dic[bigram]:

                if comments_df['month'][i] == month and bigram in comment_text:
                    bigram_l.append(bigram)
                    month_l.append(month)
                    video_id_l.append(comments_df['c_video_id'][i])
```

```
            if comments_df['month'][i] == month and bigram in comment_text:
                bigram_l.append(bigram)
                month_l.append(month)
                video_id_l.append(comments_df['c_video_id'][i])

# Step 2
# check the video text for each video, see if it contains the bigram
# save the bigram name, month, and video id

for i,video_text in enumerate(videos_df['text_string']):

    for bigram in bigram_months_dic:

        for month in bigram_months_dic[bigram]:

            if videos_df['month'][i] == month and bigram in video_text:
                bigram_l.append(bigram)
                month_l.append(month)
                video_id_l.append(videos_df['video_id'][i])

# Step 3
# create a dataframe and save the above information
ref_videos_dic = {'bigram': bigram_l,
                  'month': month_l,
                  'video_id': video_id_l,
                  }
ref_videos_df = pd.DataFrame(ref_videos_dic)

# add a column shows how many times the bigram is mentioned in the comments for each video
ref_videos_df = ref_videos_df.value_counts().reset_index(name='bigram_mentioned_times')
ref_videos_df
```

Bigram Trend Analysis (Code) cont. 8

```
# Step 4
# join the previous generated dataframe with some column in the video dataframe by video id
ref_videos_df = pd.merge(ref_videos_df, videos_df[['video_id','likes_count','comments_count','video_url','video_desc','video_transcript']], on="video_id

# drop the video id column (not useful)
ref_videos_df = ref_videos_df.drop(columns=['video_id'])

# sort by bigram name, month, and the times that bigram is mentioned in the comments
ref_videos_df = ref_videos_df.sort_values(by=['bigram', 'month','bigram_mentioned_times'], ascending = [True, True, False], ignore_index=True)

return ref_videos_df
```

```
# get the videos reference data frame for the peaked bigrams
peaked_videos_df = getVideoRefDF(videos_df, comments_df, bigram_and_peaked_months)
peaked_videos_df
```

```
# save to local folder
peaked_videos_df.to_excel('data/peaked_bigrams_video_reference.xlsx', index=False)
```

Color Bigram Trend Analysis (Code)

```
# Open color bigram file
df_color = pd.read_excel('data/color_bigrams_for_trend_analysis.xlsx')

# group data frame by month and bigrams
df_color_grouped = df_color.groupby(['month','bigram'])['bigram'].count().unstack()

# replace the null values with 0
df_color_grouped.replace(np.nan, '0',inplace=True)
df_color_grouped = df_color_grouped.astype(int)
df_color_grouped
```

```
# an overview of the graph of all color bigrams
# fig,ax = plt.subplots(figsize=(15,9))
df_color_grouped.plot(figsize=(15,9))
plt.xticks(np.arange(len(df_grouped.index)), df_grouped.index, rotation=45)
plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
```

Get Video Links of Colored Bigrams: if the bigram is mentioned more than the threshold value in a month, we provide links to those videos.

```
# you can change the threshold
threshold = 10
```

```
# get the color bigrams mentioned more than the threshold and the months

color_bigram_and_months = {}

for color_bigram in df_color_grouped.columns:

    months = []
    for i, mentioned_times in enumerate(df_color_grouped[color_bigram]):
        if mentioned_times > threshold:
            months.append(df_color_grouped.index[i])

    if months: # if the list is not null
        color_bigram_and_months[color_bigram] = months

color_bigram_and_months
```

```
{'blue plate': ['2022-05'], 'pewter green': ['2022-02']}
```

Color Bigram Trend Analysis (Code) cont. 2

```
# get the video reference data frame of color bigrams
color_videos_df = getVideoRefDF(videos_df, comments_df, color_bigram_and_months)
color_videos_df
```

	bigram	month	bigram_mentioned_times	likes_count	comments_count	video_url	video_desc	video_transcript
0	blue plate	2022-05	20	8208	168	https://www.tiktok.com/@mississippi_kween/vide...	#tartarsauce #tartar #sauce #homemade #cooktok...	
1	pewter green	2022-02	9	35700	559	https://www.tiktok.com/@yourlifeiswhatyoumakei...	I dub this the #paintstickchallenge #paintsti...	
2	pewter green	2022-02	4	8823	508	https://www.tiktok.com/@yourlifeiswhatyoumakei...	DIY kitchen makeover #diy #diyer #diykitchen #...	wait wait wait

```
# save to local folder
color_videos_df.to_excel('data/color_bigrams_video_reference.xlsx', index=False)
```

END.