Introduction

Influencer and Brand Dataset

This dataset is provided by Seungbae Kim, an Assistant Professor at the University of South Florida

https://sites.google.com/site/sbkimcv/dataset/instagram-influencer-dataset?authuser=0

This dataset contains 1.6 M Instagram posts that mention 26,910 brand names and were published by 38,113 influencers. There are two types of brand mentions in influencer marketing, including sponsored brand mentions and non-sponsored brand mentions. If an influencer gets paid by posting advertising posts that mention the name of the brand, then that is considered a sponsored post. There are JSON files of the posts and their sponsorship label.

Posts are labeled as 'Sponsored' if the post either uses the branded content tool or contains sponsorship-related hashtags (e.g., #ad, #sponsored, #paidAd). The details of the data collection method and labeling rules are described in the paper "Discovering Undisclosed Paid Partnership on Social Media via Aspect-Attentive Sponsored Post Learning" published in WSDM '21.

Data Structure

1. **post_info.txt**

- This file contains a list of 1,601,074 posts. Each line represents a post and is composed of 4 columns of information.

[Post ID] [USER name] [Sponsorship label] [JSON file] [Image files]

- The first column is **post ID** which starts from 0 and ends with 1,601,073
- The second column is the **Instagram USER name** of the influencer who publishes the corresponding post
- The third column is **the sponsorship label**. The value is 1 if the post is a sponsored post, otherwise 0
- The fourth column is the name of the **corresponding JSON file**

- The last column is the name(s) of the **corresponding image file(s)**. Note that a single post can have multiple image files.

2. **json_files.zip**

- This zip file contains JSON files of the 1,601,074 posts.
  JSON files have various information such as captions, likes, comments, timestamps, sponsorship, usertags, etc

3. **profiles_influencers.zip**

- This zip file contains Instagram profiles of the 38,113 influencers.
- The name of each file indicates the username of the corresponding influencer.
- When you open up the file using text editors, you will see one line of following information separated by Tab.

[Name] [Followers] [Followees] [Posts] [URL] [T/F] [Category] [Bio] [E-mail] [Phone] [Profile_pic]

4. **profiles_brands.zip**

- This zip file contains Instagram profiles of the 25,282 brands.
- Note that profiles of 1,628 brands are not available. (Total brands: 26,910)
- The fields of each profile are the same as influencer profiles
- Here is all the available data:

Posts (JSON)	Influencer Profile	Brand Profile
__typename caption_is_edited comments_disabled edge_media_preview_like edge_media_to_caption edge_media_to_comment edge_media_to_sponsor_user edge_media_to_tagged_user gating_info has_ranked_comments id is_ad is_video	Influencer_Name Influencer_Followers Influencer_Followees Influencer_Posts Influencer_URL Influencer_T/F Influencer_Category Influencer_Bio Influencer_E-mail Influencer_Phone Influencer_Profile_pic Influencer_username	Brand_Name Brand_Followers Brand_Followees Brand_Posts Brand_URL Brand_T/F Brand_Category Brand_Bio Brand_E-mail Brand_Phone Brand_Profile_pic Brand_username

location media_preview shortcode should_log_client_event taken_at_timestamp tracking_token viewer_can_reshare viewer_has_liked viewer_has_saved viewer_has_saved_to_collection viewer_in_photo_of_you owner_full_name owner_username owner_id		

Methodology

Influencer and Brand Dataset:

1. Reading Data

Challenge: We needed to process 3 folders with multiple files. Brand and Influencer folders contain separate files for each profile. JSON folder contains a separate JSON file for each post.

Solution:

1. Read profile data from all profile files (Brand Profiles and Influencer Profiles)

In [12]:
```
#get all the files in the users_brands_SPOD and users_influencers_SPOD folders
#read_profile function appends all the data and returns a Dataframe

profile_brands = read_profile(glob.glob('users_brands_SPOD//*')) # get
profile_influencers = read_profile(glob.glob('users_influencers_SPOD//*'))

#save the processed data to csv files for simplicity and time management
# read_profile is a heavey opration and must be avoided whenever we can
profile_brands.to_csv('profile_brands.csv',index=False)
profile_influencers.to_csv('profile_influencers.csv',index=False)

# you should keep this block commented out once you have full data in the csv file
# this block should only be uncommented if you have new profile data to add  and it should be
```

In [8]:
```
profile_brands
```

Out[8]:

	Brand_Name	Brand_Followers	Brand_Followees	Brand_Posts	Brand_URL	Brand_T/F	Brand_Category	
0	Joerg Koch/ 032c	201421.0	496.0	1513.0	http://www.032c.com/store	True	Local Events	#032c Summ BIG FLA
1	06 Milano	5781.0	206.0	852.0	http://www.06milano.com/	True	Personal Goods & General Merchandise Stores	06MILANO shoes- bags [
2	080 Barcelona Fashion	46484.0	372.0	3435.0	https://bit.ly/2JB40G4	True	Non-Profits & Religious Organizations	Instagra Barce
3	KHALFAN MOHAMMED AL SUWAIDI	70049.0	1593.0	356.0	NaN	False	NaN	
4	100	1046673.0	187.0	51.0	http://cargocollective.com/fon	True	Personal Goods & General Merchandise Stores	Every Post
...	...	...	...	...	...	...	...	
25276	TAKOI	20213.0	872.0	1508.0	https://www.framehazelpark.com/experience/magn...	True	Restaurants	dinner, drin patio snac
25277	Timini Egbuson	111057.0	2961.0	364.0	https://bit.ly/2D4lTep	True	Creators & Celebrities	film|Influence|I
25278	Franziska	26619.0	991.0	1014.0	NaN	False	NaN	If you hat INTER

In [48]:
```
profile_influencers
```

Out[48]:

nfluencer_Name	Influencer_Followers	Influencer_Followees	Influencer_Posts	Influencer_URL	Influencer_T/F	Influencer_Category
NaN	2031	53	2520	https://weibo.com/u/3912097415	False	NaN
Rocketgirl	6453	710	1059	https://imgur.com/a/AZW18	False	NaN
미영	31521	2202	601	https://www.youtube.com/channel/UC6LKoR7PTidlb...	False	NaN
Kris Giles	12630	7069	261	http://modelmayhem.com/	True	Creators & Celebrities
Vincent 🎅 & Eliott 🐱	18583	523	487	NaN	True	Creators & Celebrities
...	...	...	...	...	...	...
𝓢𝓪𝓻𝓪	22165	194	490	http://item.woomy.me/Instagrammer/____sarang	True	Creators & Celebrities
Sebastian	4286	1279	1017	NaN	True	Creators & Celebrities
Aline	35204	492	1202	NaN	True	Creators & Celebrities
絵梨佳	16566	514	418	https://pressblog.me/users/_____erikaitou___	True	Creators & Celebrities

2. Save the processed data (Dataframes) to csv files for simplicity and time management
3. Read post_info.txt file, map each field to its respective header
4. Process raw JSON files and normalize the data
5. Load JSON data into a dataframe. Save the dataframe to csv file for simplicity and time management

In [5]:
```
def process_json(data):
    """

    a function to process the raw json data and normalize the data


    :param data:
    :return:
    """

    dataset = {}
    dataset['edge_media_to_tagged_user'] = process__User_nodes(data.get('edge_media_to_tagged_user',{})) # get a comma
    dataset['edge_media_to_sponsor_user'] = process__User_nodes(data.get('edge_media_to_sponsor_user',{})) # get a comm
    dataset['edge_media_to_caption'] = get_caption(data.get('edge_media_to_caption',{})) # get the post caption text
    dataset['owner_full_name'] = data.get('owner',{}).get("full_name",None) # get post owners  full name if avilable, D
    dataset['owner_username'] = data.get('owner',{}).get("username",None) # get post owners  username if avilable, Defa
    dataset['owner_id'] = data.get('owner',{}).get("id",None) # get post owners id if avilable, Defaults to None
    dataset['edge_media_preview_like'] = data.get('edge_media_preview_like',{}).get("count",None)  # get total likes or
    dataset['edge_media_to_comment'] = data.get('edge_media_to_comment',{}).get("count",None) # get the total comment c
    dataset["Id"] = data.get("id",None) # get the post id if avilable, Defaults to None
    dataset["Is_ad"] = data.get("is_ad",False)  # get the is_ad value  if avilable, Defaults to False
    dataset["Is_video"] = data.get("is_video",False)  # get the is_video value if avilable, Defaults to False
    dataset["Location"] = data.get("location",None)  # get the location data if avilable, Defaults to None
    dataset["json_file"] = data.get("id","")+'.json'  # get the json_file name json_file = [postid].json
    return dataset
```

In [35]:
```
chunk_df
```

Out[35]:

edge_media_to_caption	owner_full_name	owner_username	owner_id	edge_media_preview_like	edge_media_to_comment	Id
Our family for @calvinklein!! #ad #MyCalvins ❤...	None	None	208560325	1313524	13702	1698217411940199075
Had the most unbelievable baby shower - we fel...	Khloé	khloekardashian	208560325	4289037	18116	1732396352945810332
#ad If you've been wondering what I've been do...	None	None	208560325	1095877	7519	1805473239544492560
Such a beautiful shoot with my beautiful famil...	Khloé	khloekardashian	208560325	1544477	4236	1836610207729726604
Wearing my #GIRLKING nail polish shades from @...	Khloé	khloekardashian	208560325	1166244	4915	1861190707437801501
#ad Progress update! Enjoying my meal replace...	Khloé	khloekardashian	208560325	1136363	11110	1871428720393579263
Girls go follow @prettylittlething for the bes...	Khloé	khloekardashian	208560325	1232514	5932	1898078487604962297
#ad So look at this you guys! The flat tummy b...	None	None	208560325	1250708	10213	1919051611767026169
In #mycalvins @calvinklein #ad	Khloé	khloekardashian	208560325	4050350	51589	1938774620551599060
	None	None	208560325	301286	1244	1939245535303266594
#ad So Mom gave the finger to dry hands with @...	Khloé	khloekardashian	208560325	1291475	7024	1957732720616652277
Loving my BoxyCharm Box. Five full sized produ...	None	None	18428658	412103	7702	1882129873196286372

2. Filtering & Matching

- Filter JSON posts by Sponsorship Label = 1 (we are interested only in Sponsored posts)
- Select brands with > 1.5M and less than 1.6M followers

In [16]:
```
profile_brands = profile_brands[profile_brands['Brand_Followers'].between(1500000,1600000)]
profile_brands
```

2836	2836	hotmiamistyles	1597934.0	1059.0	10842.0	http://www.HotMiamiStyles.com/	True	Personal Goods & General Merchandise Stores	The
3147	3147	Shawn Johnson East	1561500.0	2732.0	2628.0	https://youtu.be/EEMmN7miHAg	True	Creators & Celebrities	Wif
3190	3190	Robbie Williams	1544725.0	40.0	1214.0	https://robbiewilliams.lnk.to/revealIN	True	Creators & Celebrities	
3209	3209	CREME PARA ESTRIAS	1525982.0	347.0	2533.0	http://www.100estrias.com.br/	True	Personal Goods & General Merchandise Stores	Mich
3387	3387	Barneys New York	1558855.0	1139.0	7673.0	https://bit.ly/2NKF4KM	True	Personal Goods & General Merchandise Stores	
4112	4112	live lokai	1585475.0	54.0	2089.0	http://lokai.co/breastcancer	True	Personal Goods & General Merchandise	FIND

- Tokenize post captions (split string by whitespace remove all special characters except @)
- Find usernames using @ in the captions

In [22]:
```
def get_all_usernames_from_tokens(edge_media_to_caption_tokes):

    # check if token array is empty or not if empty  return null
    if len(edge_media_to_caption_tokes) == 0:
        return None


    matched_brands = []  #to hold matched brands

    #itarate through all the tokens
    for token in edge_media_to_caption_tokes:
        b = str(token).strip().lower() #convert each token to lowercase and strip whitespaces for a st
        if b.startswith('@'): # if a token starts with "@" we can  safely assume its a username
            matched_brands.append(token) # add the token to the matched list

    if len(matched_brands) == 0:
        # there is no matched brands then return None
        return None
    else:
        #remove duplicates and return matched list
        return list(set(matched_brands))

def tokenize_str(string):
    #split string by whitespace remove all special chars Except
    string = str(string).lower() # convert the text into lowercase for easier matching
    string = string.replace('#'," #").replace('\n'," \n") # replace all "#" values with " #" (whitespa
    return re.sub("[^\w@ ]", "", string).split()  #split string by whitespace remove all special chars
```

- Check if the username from captions is in the list of target brands usernames using .isin

In [23]:
```
chunk_df['edge_media_to_caption_tokes'] =  chunk_df['edge_media_to_caption'].apply(tokenize_str,lambda x:x)  # tokar
```

In [24]:
```
chunk_df['brand_matched'] = chunk_df['edge_media_to_caption_tokes'].apply(lambda x:get_all_usernames_from_tokens(x))
```

In [25]:
```
chunk_df = chunk_df.explode('brand_matched')
```

In [26]:
```
chunk_df.dropna(subset=['brand_matched'],inplace=True) #remove all rows where brand_matched is null
del chunk_df['edge_media_to_caption_tokes'] # delete edge_media_to_caption_tokes - > we dont need this anymore


chunk_df = chunk_df[chunk_df['brand_matched'].isin(profile_brands['Brand_username'])] #check if the matched brand is
```

- Get 20 top Influencers (by following) for each caption/brand username match

Step4. Group by brand Names and keep columns with attributes useful for KG

```
In [32]: g = df.groupby(['brand_matched']).apply(lambda x: x.nlargest(20,['Influencer_Followers'])).reset_index(drop=True)
```

3. Join data

Challenge: we need to combine all of our dataframes to start adding additional node attributes and build Nodes and Relationships files for Neo4j in the next steps

Solution:
We joined all 3 tables (posts, Influencers, Brands) and selected columns that provide valuable information about objects (nodes) and relationships.

- Join Posts dataframe with brand matches with Profile Brands dataframe ('brand_matched' from the new df = 'Brand_username' in the Profile Brands Dataframe)
- Join Posts dataframe with brand matches with Profile Influencers dataframe ('username' from the new df = 'Influencer_username' in the Profile Influencers Dataframe)

```
In [27]: # join the new dataframe with profile_brands Dataframe,  keep all records where 'brand_matched' from the new df  equals
         chunk_df = chunk_df.merge(profile_brands,left_on='brand_matched',right_on='Brand_username',how='left')

         # join the new dataframe with profile_influencers Dataframe,  keep all records where 'username' from the new df  equals
         chunk_df = chunk_df.merge(profile_influencers,left_on='username',right_on='Influencer_username',how='left')
```

Our resulting table is a joined table featuring:

- Brands of interest and their attributes (# of followers, # of followees, category etc.)
- Posts that mention Brands of interest in captions + their attributes (# of likes, # of comments)
- Influencer who published those posts and their attributes (# of followers, # of followees, category etc.)

Out[35]:

brand_id	Brand_Name	Brand_Followers	Brand_Followees	Brand_Posts	Brand_URL	Brand_Category	post_id	
3209	CREME PARA ESTRIAS	1525982.0	347.0	2533.0	http://www.100estrias.com.br/	Personal Goods & General Merchandise Stores	1877001634463122445	Eu e meu
3209	CREME PARA ESTRIAS	1525982.0	347.0	2533.0	http://www.100estrias.com.br/	Personal Goods & General Merchandise Stores	1891646163346665904	Minha cara me
3209	CREME PARA ESTRIAS	1525982.0	347.0	2533.0	http://www.100estrias.com.br/	Personal Goods & General Merchandise Stores	1916289352699275991	Tô de olho e
3209	CREME PARA ESTRIAS	1525982.0	347.0	2533.0	http://www.100estrias.com.br/	Personal Goods & General Merchandise Stores	1936582221779537401	F comp
3209	CREME PARA ESTRIAS	1525982.0	347.0	2533.0	http://www.100estrias.com.br/	Personal Goods & General Merchandise Stores	1961799656992243034	Minha @ofantastic @100est

Wikipedia API Dataset

To expand our knowledge and information about the brands we are interested in, we want to scrape the brand's wikipedia page (if they have a wikipedia page) and extract keywords from the summary attribute. To achieve this, we used the wikipedia search API and the wikipedia python package. The search API allowed us to query brand names for all wikipedia pages that match our query. We then used the wikipedia python API package to extract keywords from the summary attribute of the wikipedia, using Lemmetization and LDA strategies.

Methodology

First, we installed all appropriate packages into our environment using the commands below:

```
pip install gensim
pip install spacy
pip install nltk
pip install py-stringmatching
pip install pyLDAvis
pip install wikipedia
pip install Unidecode
```

Next we cleaned the extracted brand names so they were able to be inputted into the json wikipedia search API:

```
]: def cleanListOfWords(listOfWords):
       listOfWords =  [remove_emoji(i).lower() for i in listOfWords if i != '']
       listOfWords = [re.sub('\S*@\S*\s?', '', word).lower() for word in listOfWords]
       listOfWords = [re.sub(r"[^a-zA-Z0-9 ]", " ", word) for word in listOfWords]
       listOfWords = [re.sub('\s+', ' ', word) for word in listOfWords]
       listOfWords = [re.sub("\'", "", word) for word in listOfWords]
       listOfWords = [re.sub(r'[^\w\s]', '', word) for word in listOfWords]
       listOfWords = [unidecode(word) for word in listOfWords]
       return listOfWords
```

Next we used our clean brand list to extract the list of potential matching wikipedia pages with the brands. For each brand, we extracted the list of wikipedia page titles. We then used qgram-tokenization to tokenize each title from our query. Next, we conducted a cosine similarity score to compare the brand name to each result. We compared each result to both the brand name and the brand name and “(company)”. We did this because exact brand name can mean different things. For example, the company Amazon can be both the company and the rainforest. To make sure the Amazon (company) wikipedia page was selected, we compared both titles. We then chose the title with the maximum similarity score.

```
]: q3_token = sm.QgramTokenizer(qval=3)
   cosine = cos.Cosine()
   jaccard = sm.Jaccard()
```

```
]: phrasesList = []
   for (idx, brand) in enumerate(cleanedBrandNames):
       #get rid of all unecessary words that could skew cosine similarity
       plainBrandName = brandNames[idx]
       phrase = '%20'.join([remove_stopwords(abst) for abst in brand.split(' ')])
       url = f'https://en.wikipedia.org/w/api.php?action=query&list=search&srsearch={phrase}&utf8=&format=json'
       jsonPhrase = pd.read_json(url, orient='records')

       scores = []

       for brandToToken in [brand + '(company)']:
           phraseToken = q3_token.tokenize(brandToToken)
           for result in jsonPhrase['query']['search']:
               title = result['title'].lower()
               q3TokenSample = q3_token.tokenize(title)
               cosScoreQ3 = cosine.get_sim_score(q3TokenSample, phraseToken)
               scores.append([result['title'], cosScoreQ3])

       if('roxy' in brand):
           scores = [["Quicksilver (company)", 1]]
       if('moda' == brand):
           scores = [["Modà", 1.1]]

       if(scores):
           maxScore = max(scores, key=lambda item:item[1])
           phrasesList.append([plainBrandName, maxScore[0]])
       else:
           phrasesList.append([plainBrandName, ''])
```

With our appropriate wikipedia titles attained, we extracted a summary of the wikipedia page with the summary function of the wikipedia API. We then split the summary into individual words and removed stop words. Next we created a bigram and trigram of the list of words. We then utilized the lemmetization and LDA model to extract topics from the list of summary words. This gave us multiple lists of topics. We then wanted to consolidate our keywords further and used the attained topics as a list of words and found the keywords of those resulting topics. This

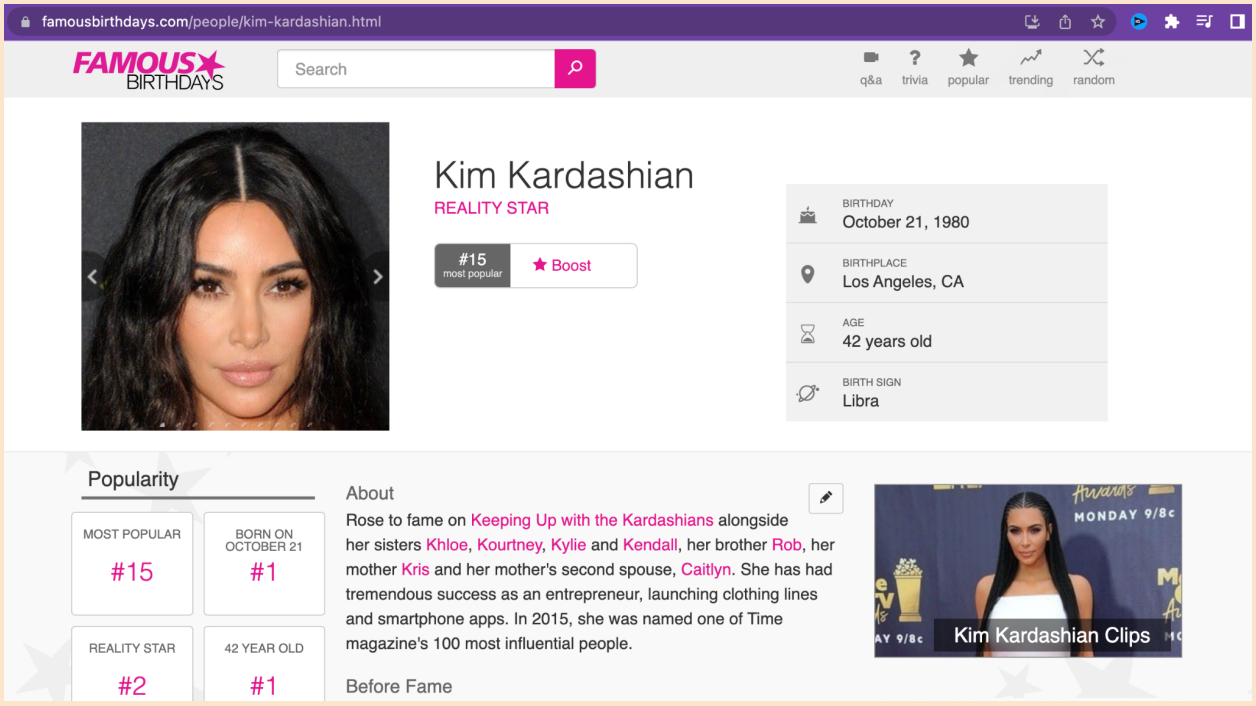
resulted in four keywords per brand. These results will be added as an attribute to the brand nodes.

```
[ ]: keyWordsList = []
     for brand in phrasesList:
         topicsOfTopics = []
         if(brand[1] != ''):
             wikiTitle = brand[1]
             wikiPage = wikipedia.summary(title = wikiTitle, auto_suggest=False).split(' ')
             topics = findKeywords(brand[0], wikiPage, 10);
             topicsOfTopics = findKeywords(brand[0], topics, 2)

         keyWordsList.append([brand[0], topicsOfTopics])

     print(keyWordsList)
```

```
def findKeywords(brandName, listOfWords, numWords):
    wikiPage = cleanListOfWords(listOfWords)
    wikiPage = [word for word in wikiPage if word.lower() not in brandName.lower().split(" ")]

    absGram = [remove_stopwords(abst) for abst in wikiPage]
    absGramSplit = [remove_stopwords(abst).split(' ') for abst in wikiPage]

    bigram = gensim.models.Phrases(absGramSplit, min_count=4, threshold=1000)
    trigram = gensim.models.Phrases(bigram[absGramSplit], threshold=100)

    # Faster way to get a sentence clubbed as a trigram/bigram
    bigram_mod = gensim.models.phrases.Phraser(bigram)
    trigram_mod = gensim.models.phrases.Phraser(trigram)

    def make_bigrams(texts):
        return [bigram_mod[doc] for doc in texts]

    def make_trigrams(texts):
        return [trigram_mod[bigram_mod[doc]] for doc in texts]

    data_words_trigrams = make_trigrams(absGramSplit)

    nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

    tri_lemmatized = lemmatization(data_words_trigrams, nlp, allowed_postags=['NOUN', 'ADJ', 'VERB'])

    id2word = corpora.Dictionary(tri_lemmatized)
    texts = tri_lemmatized
    corpus = [id2word.doc2bow(text) for text in texts]

    allTopics = []
    ldaModel = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                               id2word=id2word,
                                               num_topics=numWords,
                                               random_state=133,
                                               update_every=10,
                                               chunksize=len(wikiPage),
                                               passes=10,
                                               alpha='auto',
                                               per_word_topics=False)

    doc_lda = ldaModel[corpus]

    topics = []

    for idx, topic in ldaModel.show_topics(formatted=False, num_words= numWords):
        topics.extend([w[0] for w in topic])

    return topics
```

FamousBirthdays.com Dataset

This website features Celebrity profiles in a simple and entertaining format that is structured and standard for each person. We supplemented the Influencer Profile attributes with data such as their Birthday, Birthplace, Age, etc. There was no API readily available so most of the data was scraped using Selenium.



Methodology

For our structured dataset, we scraped the data from the famousbirthdays.com website that contains a wide range of “famous” people ranging from celebrities with over 1 million followers to more niche influencers with less than 10,000 followers on instagram.

Problems and Concerns

There was no API for this website, so that was one of our earlier concerns. Initially, we wanted to find a way to grab all possible celebrities from the site and fuzzy match their name to the name in the influencer names attributes. In addition, there was no master list of celebrities you could sort by category. From the site you’re able to select people by a category such as birthday, birthplace, TV shows and more, but even then only the top 48 celebrities were displayed on the list. Unfortunately, there was no option to go onto the second page so we were limited to at most 48 people in any one category.

Resolutions

We ended up using Selenium to automate the clicks and search features on the website in order to scrape the data from the stat box. Initially we wanted to find a way to grab all possible celebrities on the site. With selenium we set it up so that it automated the Random button and scraped every possible person, but you start to notice repeated information and it was not going to be the most efficient use of our time.

We also tried to scrape as many people as possible in a short amount of time, but the page load was too fast. We were met with CAPTCHAs that required a human to answer and Selenium was not built for that kind of automation. Our solution to that was to have it sleep for a few seconds in between searches. Using the sleep function without any specific condition defeats the purpose of automation, so in the future we'd like to explore more automation techniques.

In the end we narrowed down the number of influencers and automated the search using selenium to input each name into the search bar then grabbing the XPath for the exact location. Good news, some people only went by their social media handles such as 'grav3yardgirl' and it was as straightforward as just entering their handle in the search bar. Some names did not give any results and we filled that in the N/A

Process for Scraping

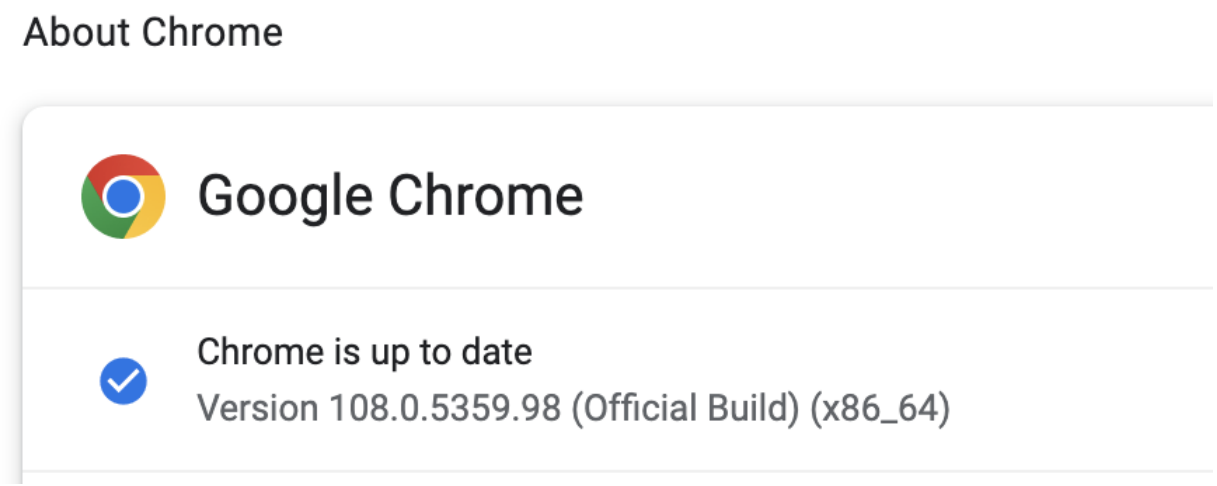
How to Set up chrome driver instructions and what you need to install in order to get it running.

Step 1 - Install Selenium and Access WebDriver using Chrome

```
pip install selenium
pip install webdriver-manager
```

The webdriver will be used to automatically open a web browser to the FamousBirthdays.com website.

- Locate which version of Chrome you have from the three vertical dots in the upper right hand corner Help > About Google Chrome
- Download the webdriver https://chromedriver.chromium.org/downloads



- Take note of where you saved your webdriver download on your local computer. Mine is just saved in the default downloads folder. Here, I kept the webdriver in the downloads folder

```
DRIVER_PATH = '/Users/squach/Downloads/chromedriver'

driver = webdriver.Chrome(options = options,
executable_path=DRIVER_PATH
```

Step 2 - Imports

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait

import time
```

Knowledge Graph

Building Steps

To build our nodes and relationships, we utilize our main dataframe, in which each record was the post and it's associated brand and influencer information. Instead of creating new ids for each entity, we utilized their given ids from the raw data source. Below is an example of the brand node creation:

Create Nodes and Relationships

Create Brand Nodes

```
[ ]: nodes = []
for word in keyWordsList:
    brand = df_final[df_final['Brand_Name'] == word[0]].iloc[0]
    nodes.append([brand['brand_id'], "Brand", brand['Brand_Name'], word[1], brand['Brand_Followers'], \
                  brand['Brand_Followees'], brand['Brand_Posts'], brand["Brand_URL"], brand["Brand_Category"]])

brandNodeDf = pd.DataFrame(nodes, columns=['brand_ID', ':LABEL','Brand_Name','Keywords', "Number_Of_Followers", "Number_Of_Followees", "
```

```
[ ]: brandNodeDf.head()
```

After creating each node, our dataframe's structure allowed us to create the edges dataframes very easy as well. Here is an example of our brand to post edge creation:

```
: relbrandtopost = []
for index, row in df_final.iterrows():
    postId = postNodesDf[(postNodesDf['Caption'] == row['post_caption']) \
                         & (postNodesDf['Number_of_Likes'] == row['post_likes'])]['post_ID'].iloc[0]
    brandId = brandNodeDf[brandNodeDf['Brand_Name'] == row['Brand_Name']]['brand_ID'].iloc[0]
    relbrandtopost.append([brandId, "SPONSORED", postId])
edge_brandtopost = pd.DataFrame(relbrandtopost, columns=['brand_ID', 'TYPE', 'post_ID'])
edge_brandtopost.head()
```

We generated 5 CSV files:

- Brand Nodes
- Influencer Nodes
- Post Nodes
- Brand - Post relationships
- Influencer - Post relationships

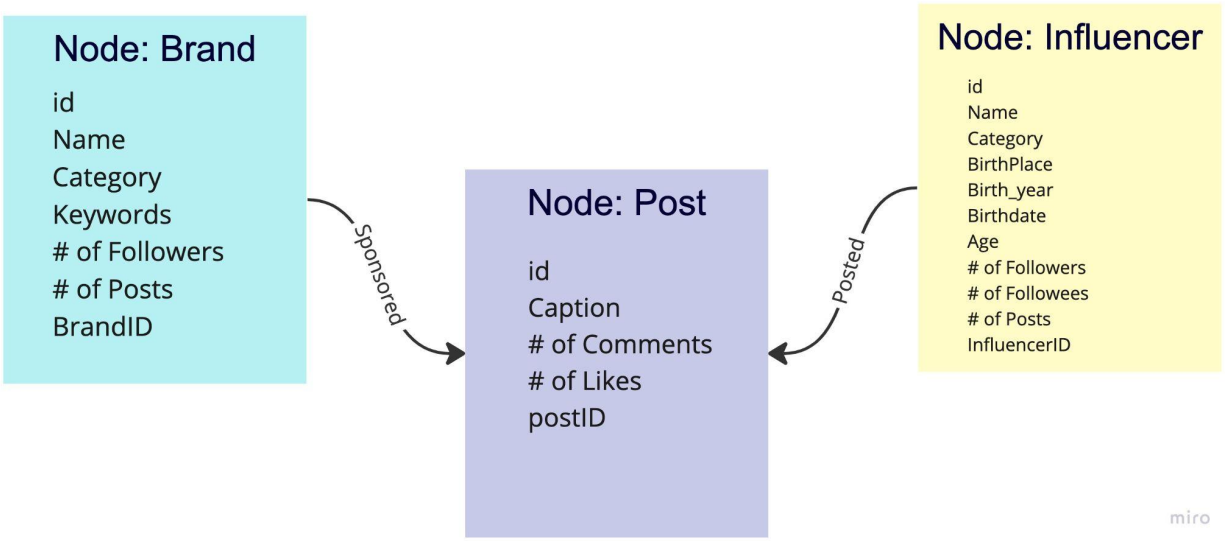
We used Neo4j browser to upload CSV files:

```
LOAD CSV WITH HEADERS FROM "file:///InfluencerNodes.csv" as row
MERGE (influencer:Influencer {influencerID: row.influencerID})
ON CREATE SET influencer.Name = row.Influencer_Name,
influencer.Followers = row.Number_Of_Followers,
influencer.Followees = row.Number_of_Followees,
influencer.Posts = row.Number_Of_Posts,
influencer.Category = row.Influencer_Category,
influencer.Birthdate = row.Birthdate,
influencer.Birth_Year = row.Birth_Year,
influencer.Age = row.Age,
influencer.BirthPlace = row.Birthplace;
```

```
LOAD CSV WITH HEADERS FROM "file:///BrandNodes.csv" as row1
MERGE (brand:Brand{brandID: row1.brand_ID})
ON CREATE SET brand.Name = row1.Brand_Name,
brand.Followers = row1.Number_Of_Followers,
brand.Followees = row1.Number_of_Followees,
brand.Posts = row1.Number_Of_Posts,
brand.Keywords = row1.Keywords,
brand.Category = row1.Categories;

LOAD CSV WITH HEADERS FROM "file:///PostNodes.csv" as row2
MERGE (post:Post{postID: row2.post_ID})
ON CREATE SET post.Likes = row2.Number_of_Likes,
post.Comments = row2.Number_of_Comments,
post.Caption = row2.Caption;

LOAD CSV WITH HEADERS FROM "file:///Edges_Brand.csv" AS row3
MATCH (brand:Brand{brandID: row3.brand_ID}),
(post:Post{postID: row3.post_ID})
CREATE (brand)-[:SPONSORED {type: row3.TYPE}]->(post)

LOAD CSV WITH HEADERS FROM "file:///Edges_Infl.csv" AS row4
MATCH (influencer:Influencer{influencerID: row4.influencer_ID}), (post:Post{postID:
row4.post_ID})
CREATE (influencer)-[:POSTED {type: row4.TYPE}]->(post)
```

KG Structure

- Nodes
  - Influencers **(279)**
  - Brands **(35)**
  - Posts **(441)**
- Edges/Relationships
  - Sponsored: Brand to Post **(442)**
  - Posted: Influencer to Post **(441)**


Node: Brand
id
Name
Category
Keywords
# of Followers
# of Posts
BrandID
Sponsored
Node: Post
id
Caption
# of Comments
# of Likes
postID
Posted
Node: Influencer
id
Name
Category
BirthPlace
Birth_year
Birthdate
Age
# of Followers
# of Followees
# of Posts
InfluencerID
miro

Sample Queries

What is the Average Age of the Sponsored Influencer by Brand?
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)
Return b.Name, avg(toInteger(i.Birth_Year)) as Birth_Year
Order by Birth_Year asc

Conclusion: the oldest Influencers are hired by Bottega Veneta (average Year of Birth is 1971) and the youngest by Gymshark Women(average Year of Birth is 1998)

Which Country "produces" the most Influencers?
Match (i:Influencer)
Return i.BirthPlace, count(i.Name) as NumberofInfluencers
Order by NumberofInfluencers desc

Conclusion: top 3 countries are US, England, and Canada

Which Influencer Category is most popular among Brands?
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)
Return i.Category as InfluencerCategory, Count(i.Category) as NumberOfCategories
order by NumberOfCategories desc

Conclusion:"Creators & Celebrities" is the most popular category

What is the average Following of Influencers by Brand? Which Brands hire Top Influencers (by Following size)
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)
Return b.Name, avg(toInteger(i.Followers)) as Following
Order by Following desc

Conclusion: Amazon, Adidas and Moda tend to hire Mega Influencers (1M+ followers)

Which brands have the highest engagement per collaboration with an influencer?
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)
return Count(distinct i) as influencerCount, Sum(DISTINCT toInteger(p.Likes)) as NumberOfLikes,
Sum(DISTINCT toFloat(p.Likes))/Count(i) as Avg_Likes, b.Name as BrandName
order by Avg_Likes desc
Conclusion: Amazon, Moda, Quay Australia, and Adidas had highest average likes per post

Does repeated collaboration with the same influencer increase or harm engagement?
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)

```
return Count(i) as NumberOfCollaborations, i.Name as InfluencerName, b.Name as
BrandName,SUM(toFloat(p.Likes))/(toFloat(i.Followers)) as ratio
order by ratio desc
```

Conclusion: Repeated posts with the same brand and influencer collaboration increased engagement substantially

Who are the most popular Influencers for Brands in Personal Goods & General Merchandise Stores Category?

```
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)
Where b.Category = "Personal Goods & General Merchandise Stores"
Return b.Category, i.Name, count(i.Name) as InfluencerofChoice
Order by InfluencerofChoice desc;
```

Conclusion: Lorena Improta is most desirable Influencer in "Personal Goods & General Merchandise Stores" Brand Category

Which Influencer generated most positive engagement for Amazon

```
Match (i:Influencer)-[:POSTED]-(p:Post)-[:SPONSORED]-(b:Brand)
Where b.Name = "Amazon"
Return i.Name, SUM(toInteger(p.Likes)) as Likes
Order by Likes Desc
```

Conclusion: Khloe Kardashian generated most Likes for Amazon

Are there Posts Sponsored by more than 1 Brand?

```
MATCH (a:Brand)-[:SPONSORED]->(p:Post)<-[:SPONSORED]-(b:Brand)
WHERE id(a) > id(b) WITH a, b,
COUNT(p) AS count
ORDER BY count
DESC RETURN a.Name, b.Name
```

Conclusion: There is one Post Sponsored by “Universal Orlando Resort” and “Universal Studios Hollywood”

Which Brands are targeting the same audience (sponsoring the same influencers)?

```
MATCH(a:Brand)-[:SPONSORED]->(p:Post)-[]-(i:Influencer)-[]-(p1:Post)<-[:SPONSORED]-(b:Br
and)
WHERE id(a) > id(b)
WITH a,b,
COUNT(i) AS count
ORDER BY count
DESC RETURN a.Name, b.Name, count
```

Conclusion: Brands demonstrating the highest affinity are “Moda” and “CREME PARA

ESTRIAS”