TSDS3751

SOCIAL MEDIA COMPUTING

Assignment Report

Trimester 2 2020/2021 (2020 RMCO)

Student Name	Student ID
Lee Xi Jie	1161204459
Oi Zhen Fan	1181300513

Table of Contents

Introduction

Social media is becoming an effective platform for the traditional companies to perform their online marketing. There are numerous companies using unique online marketing campaigns to attract the customers, increasing popularities and their engagement rate. Therefore, figuring out the social media campaign is one of the key points of success. There are two brands of companies that we choose to compare and study in this Assignment are Under Armour and Adidas. Both of the brands data are collected and analyzed from twitter platform.

Campaign of Social Media Channel

There are various companies out there in the world that use the social media campaign as their online marketing strategy. Hence, multiple social media campaigns are being used by the companies to boost their popularity, sales etc. From our studies, there are some social media campaigns that we found and suitable for the social media channel of the sports to increase their popularity such as giveaways, contests, stories and collaborations.

Organizing a giveaway on Twitter is fast, cost-effective to engage the audience. For instance, the organizer can request the audience to follow their twitter account, liking a particular tweet, retweeting with a designated hashtag. Besides giveaway, some of the brands tend to generate promo codes for which give discounts when customers purchase their products with the promo code.

Besides, contests also a social media campaign to support the giveaway. To illustrate this point, contests can be used to stoke interest by allowing the audiences to decide between some of options. For example, when conducting a giveaway, audiences could be in a contest to figure which of the cities will be delivered to the next. Besides, sports brands can advertise through sporting events such as the super bowl, football leagues and so on.

Furthermore, posting stories is also the popular social media campaign for the brands to increase their engagement rate. For example, the brands can tweet a new collection of their product or a store launch. Moreover, some of the brands will tend to focus on the artists which collaborate with them. For instance, the artists will share their greetings which are received from the brands in their stories once they get the gift from the brands to increase the engagement rate between the artist and the brands.

Last but not least, collaborations also play a vital role in a social media channel to boost their engagement. For example, the brands can collaborate with Olympics technology to produce a product. For instance, Nike has collaborated with Apple to produce the Apple Watch Nike.

Channel Analysis

Adidas engages all the branches under himself in the twitters. For example, it works especially well for sports, influencers etc. Adidas tends to support #BlackLiveMatter which takes a strong stand against racism. In Adidas twitter account, it pins the #BlackLivesMatter in its timeline and appreciate that the success of Adidas would be nothing without Black athletes, artists etc to reduce racism. Moreover, Adidas also takes the chance to get attention from Nike.

Figure 3-1 Hashtags of BlackLiveMatter

Besides, Adidas also makes uses of the collaboration campaigns. For example, Adidas promotes their products, athletes and the events at the same time.

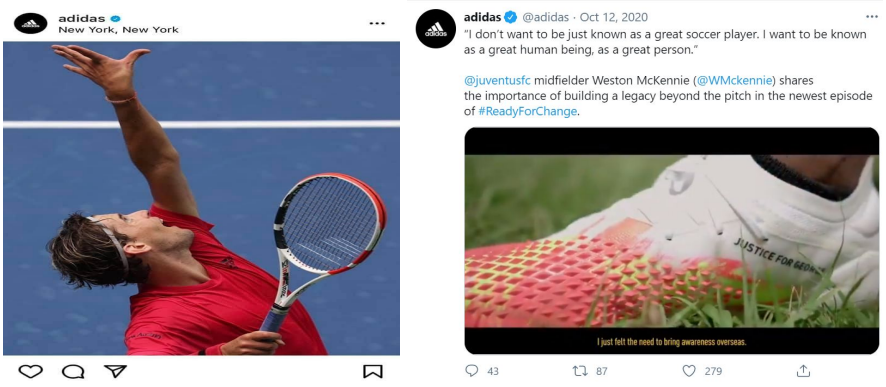

Figure 3-2 Collaboration with athletes

Furthermore, Adidas also make uses of the brand values hashtags like #ULTRABOOST to promote their products as shown in Figure 3-3.

Figure 3-3 Hashtag of Adidas product

UnderArmour makes use of special dates or events to promote their products. For example, UnderArmour carried out promotional events during 11.11 and 12.12. UnderArmour also added CNY collection to its store to promote sales during the Chinese New Year.

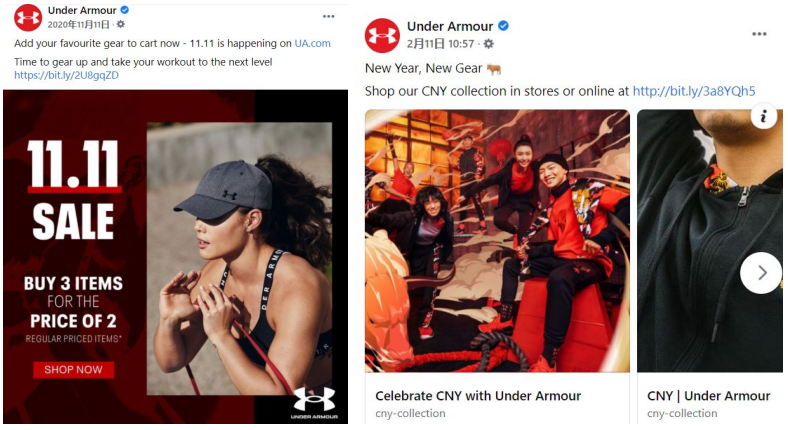

Figure 3-4 Sales promotions UnderArmour

Moreover, UnderArmour also sponsors famous athletes in order to promote their products. For example, sponsored athletes will wear their shoes to help promote sales as shown in Figure 3-5.

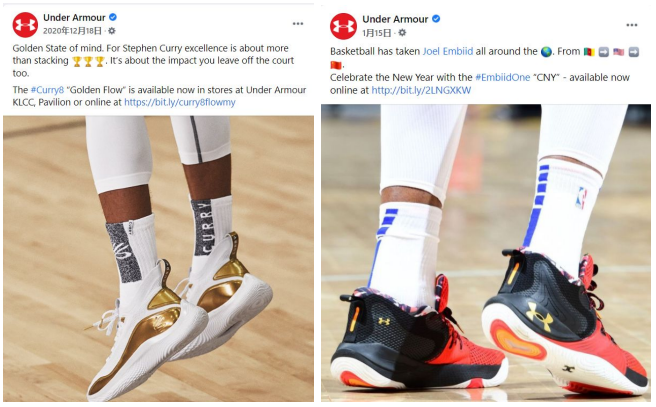

Figure 3-5 Collaboration with athlete

UnderArmour also introduced their workout series and additionally, they also promoted their sales by providing a promo code for customers to buy their products with a discount.

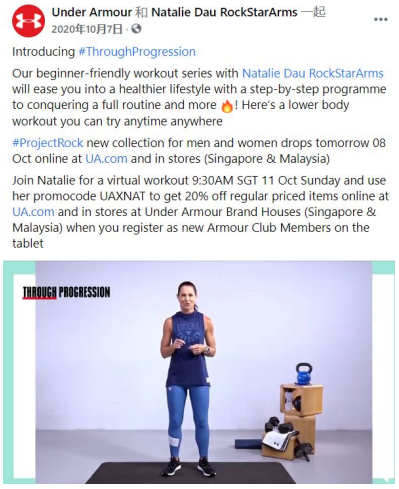

Figure 3- 6 Promo Code UnderArmour

Metrics

1. Followers Growth over the two weeks

   In this metric, we are collecting the data for the brands chosen. For example, we insert the data that we collected from twitter into the data frame and save it as a csv file named as follower_brand.csv. The data required for this metric are the Date and the Followers_Count from the brands. For this metric, we have to collect data from daily for the continuous 14 days. After collecting the data, we plot the line graph to interpret the growth of the followers in the range of time.

2. Top Hashtags based on the User Timeline

   In this metric, we are collecting the one year user timeline data from the brands chosen. After collecting the data, we saved it as a json file. Next, we count the frequency of the hashtags and analyze what designated hashtags are the most popular in the Year 2020 from the json file that we collected from twitter. For the plotting, we used the bar chart to identify which designated hashtags are the most popular at the time.

3. Top Mentioned based on the User Timeline

   In this metric, we are collecting the one year user timeline data from the brands chosen. After collecting the data, we saved it as a json file. Next, we count the frequency of the screen name that is mentioned by the brands and analyze who are the top mentioned in the year 2020 from the json file we collected from twitter. For the plotting, we used the bar chart to identify who are the most frequent users.

4. Most Active Time

   In this metric, we also use the one year user timeline data from the brands chosen. The aim for this metric is to determine which of the months is the most popular time for the brand in 2020. As the data we needed to retrieve out from the timeline are Favorite_Count, Retweet_Count based on the tweet they

posted. Nevertheless, by plotting the line graph we can see the distribution of the active time by the retweet count and favorite count of the particular brand.

5. Content Performance Analysis

   In this metric, we use the one year user timeline data and retrieve out the Status_Count, Favourite_Count, Retweets_Count from a particular brand. Besides, we have also analysed the Favorite_Count and Retweets_Count by the average. For this metric the main purpose is for us to see the content performance between the two brands. So by plotting the radar graph we can clearly see their performance in 2020.

6. Top Influencer

   In this metric, we use the search query to determine who is the Top Influencer that mentioned the brand most around the two week time. So we collected the data day by day and saved it in a json file. For this metric, we collect the data for 2 weeks which start from 26 January to 8 Feb 2021. Besides, we plot the bar chart to determine the top influencer by the frequency counts of the screen name from the data we obtained.

7. Network Density

   In this metric, we will calculate the density of the network graph for the brand chosen. We will randomly choose the 1000 followers and friends from the brands and start collecting their details. While collecting we have set the rules in the script that write the only top 5 reciprocal friends into the csv file which is named as ReciprocalFriendBrand.csv. After collecting, we calculate the network density in the beginning. The purpose of calculating the density is to determine how the connected nodes relate to each other.

8. Largest and Smallest Subgraph of the Network Graph

   In this metric, we are using the script to compute the largest and smallest subgraph for the brand that we chose. Besides, we also listed out who's in the smallest and largest subgraphs.

9. Top Betweenness Centrality

   In this metric, we will compute the user ID, degree and betweenness from the network graph.

10. Top Eigenvector Centrality

    In this metric, we will compute the user ID, degree and eigenvector from the network graph.

11. Top Closeness Centrality

    In this metric, we will compute the user ID, degree and closeness from the network graph.

Methodology

As we mentioned in the previous section, we have collected the one year data from 2020-2021 for the metrics, Content Performance, Top Hashtags, Top Mentioned, Most Active Time and two weeks data for the metrics Top Influencer and Followers Growth. Besides in the Assignment2, we have collected the random 1000 followers and friends respectively for us to construct the network graph. In this methodology section, we will explain more clearly about our step to collect and analyse the data.

For the follower growth, as we mentioned previously, we collected the Date and the Followers Count and saved it into the csv file. After collecting the data, we will obtain the csv file with 2 weeks data as shown in figure below. We have done some cleansing for our data frame because of the unknown column 0 when plotting the graph.

```
#Here is write the date and the followers count into the csv
#In our case, we saved it as follower.csv
#In this follower.csv, it consists of 2 week grwoth of followers of a particular brand
from csv import writer

def append_list_as_row(file_name, list_of_elem):
    with open(file_name, 'a+', newline='') as write_obj:
        csv_writer = writer(write_obj)
        csv_writer.writerow(list_of_elem)

username = api.get_user('Adidas')
today = date.today()
print("Today's date:", today)
follower_count = username.followers_count
print("Follower count:", follower_count)
row_contents = ['0', today, follower_count]
append_list_as_row('follower.csv', row_contents)
```

```
Today's date: 2021-02-09
Follower count: 3858674
```

	Date	Follower
0	25/1/2021	3851934
0	26/1/2021	3852331
0	27/1/2021	3852735
0	28/1/2021	3852997
0	29/1/2021	3853304
0	30/1/2021	3853851
0	31/1/2021	3854094
0	1/2/2021	3854465
0	2/2/2021	3855043
0	3/2/2021	3855627

Figure 5-1 Data collected for followers growth

Besides we collect the one year user timeline data from twitter to analyze the top mentioned, top hashtags, average content performance and most active time period from the brand chosen.

```
#In this section, we create code to collect the one year data of the user time line of the particular brnad
#So we have set the datetime starting from first day until last day of 2020
#The purpose of collecting the one year data is to determine some of the metrics such as Top Influencer,
#Top Hashtags, Top Mentiones and the Active Time Period of the particular brand
#Besides, we also create the code to determine the statistic of the content for the particular brand
def getUserTimelineOneYear(screen_name):
    startDate = datetime.datetime(2020, 1, 1)
    endDate =   datetime.datetime(2021, 1, 1)
    dirname = '/output/'
    os.makedirs(dirname, mode=0o755, exist_ok=True)
    fname = "output/user_timeline_{}_between_2020-2021.json".format(screen_name)
    print("Collecting home timeline statuses for", screen_name)
    count = 0
    with open(fname, 'w') as f:
        for page in Cursor(api.user_timeline, screen_name=screen_name, count=200).pages(20):
            for status in page:
                if status.created_at > startDate and status.created_at < endDate:
                    f.write(json.dumps(status._json)+"\n")
                    count+=1
    print("Total Tweets Collected : ",count)
    print("Tweets Collected Succesfully")

getUserTimelineOneYear('Adidas')
```

Figure 5-2 Data collected for one year user timeline

In Figure 5-2, we set the start date and the end date for tweepy to collect the tweets. We save the file as the name user_timeline_brand_between_2020-2021 in json format. Besides, the if else conditional statement in our function is to ensure the tweets are collected in the range of date time set.

```
def get_hashtags(tweet):
    entities = tweet.get('entities', {})
    hashtags = entities.get('hashtags', [])
    return [tag['text'].lower() for tag in hashtags]

t = []
co = []


def getHashes(screen_name):
    if __name__ == '__main__':
        #Jan-Feb Data
        #fname = "output/user_timeline_{}_between_1JanTo1Feb.json".format(screen_name)
        #fname = "output/user_home_timeline_{}_between_1JanTo1Feb.json".format(screen_name)
        #One year Data
        fname = "output/user_timeline_{}_between_2020-2021.json".format(screen_name)
        with open(fname, 'r') as f:
            hashtags = Counter()
            for line in f:
                tweet = json.loads(line)
                hashtags_in_tweet = get_hashtags(tweet)
                hashtags.update(hashtags_in_tweet)
            for tag, count in hashtags.most_common(10):
                t.append(tag)
                co.append(count)
                #print("{}: {}".format(tag, count))
```

Figure 5-3 Get hashes

In Figure 5-3, we create the array first. Next, we read the json file and create a counter function to retrieve the hashtags from the tweets. At the end, from the array we can plot the bar chart to determine the top hashes.

```
def get_mentions(tweet):
    entities = tweet.get('entities', {})
    mentions = entities.get('user_mentions', [])
    return [tag['screen_name'] for tag in mentions]

us = []
co = []

def getMentiones(screen_name):
    if __name__ == '__main__':
        #fname = "output/user_timeline_{}_between_1JanTo1Feb.json".format(screen_name)
        #fname = "output/user_home_timeline_{}_between_1JanTo1Feb.json".format(screen_name)
        #One year data
        fname = "output/user_timeline_{}_between_2020-2021.json".format(screen_name)

        with open(fname, 'r') as f:
            users = Counter()
            for line in f:
                tweet = json.loads(line)
                mentions_in_tweet = get_mentions(tweet)
                users.update(mentions_in_tweet)
            for user, count in users.most_common(10):
                us.append(user)
                co.append(count)
                #print("{}: {}".format(user, count))
```

Figure 5-4 Get mentioned

Besides, we use the script in Figure 5-4 to get the Top Mentioned based on the user timeline and the top influencer. Same goes to the top mentioned, we created the array for the mentioned to retrieve the frequency count and the screen name of the users.

The difference between the top influencers is we use the search query instead of the json file that consists of one year data.

```
#For the top influencer, we collect the two weeks data that whose @Adidas the most
def getquery(searchQuery,date = '2021-01-25'):
  dirname = '/output/'
  os.makedirs(dirname, mode=0o755, exist_ok=True)
  counter=0
  with open(dirname + 'searchQuery_'+searchQuery+'_before'+'_'+date+'.json','w') as f:
    for page in Cursor(api.search,q = searchQuery, until = date,count=200).pages(10):
      for status in page:
        f.write(json.dumps(status._json)+"\n")
        counter+=1
  print("Total %d searchQuery Tweets collected" % counter)

#In this for loop, we determine the date we want to collect
for i in range(25,31):
  date = '2021-01-'+str(i)
  print(date)
  getquery("Adidas",date=date)
```

Figure 5-5 Search live time data

In Figure 5-5, we put the search and the date as the parameter. Hence, tweepy api will search based on the input we inserted. For the data we collected, we save it as a json file daily. After finishing collecting, we combine the json file manually and start to retrieve the screen name for the top influencer.

```
def getActiveHomeTimeLine(period, screen_name):
    #One month data
    #file = 'output/user_home_timeline_{}_between_1JanTo1Feb.json'.format(screen_name)
    #file = 'output/user_timeline_{}_between_1JanTo1Feb.json'.format(screen_name)
    #One year data
    file = 'output/user_timeline_{}_between_2020-2021.json'.format(screen_name)
    dic = {"date":[],"favorite_count":[], "retweet_count":[]}
    with open(file) as f:
        for line in f:
            tweet = json.loads(line)
            favCount = tweet['favorite_count']
            retCount = tweet['retweet_count']
            d = tweet['created_at'].split(' ')
            #print(d)
            if period=='daily':
                d = d[2]+d[1]+d[5]
            elif period=='monthly':
                d = d[1]+d[5]
            elif period=='yearly':
                d = d[5]
            if d not in dic['date']:
                dic['date'].append(d)
                dic['favorite_count'].append(favCount)
                dic['retweet_count'].append(retCount)
            else:
                idx = dic['date'].index(d)
                dic['favorite_count'][idx] += favCount
                dic['retweet_count'][idx] += retCount

    df = pd.DataFrame(dic)
    df = df.iloc[::-1]
    plt.figure(figsize=(20,8))
    plt.plot(df['date'], df['favorite_count'], label = "Favorite", color = 'red')
    plt.bar(df['date'], df['retweet_count'], label = "Retweet", color = 'blue')
    plt.legend()
    plt.xlabel('Month', fontsize = 15)
    plt.ylabel('Count', fontsize = 15)
    plt.title('Most Popular Month of Adidas', fontsize = 20)
    plt.xticks(rotation = 'horizontal')

getActiveHomeTimeLine('monthly',"Adidas")
```

Figure 5-6 Most active time

In order for us to get the most active time for the brands, we used the one year data to determine the metric. We used the dictionaries to split the date time of the tweets. After splitting the data, we append the counts into the dictionaries in a for loop manner. After computation, the frequency of the favourite count and retweet count will be saved into the data frame.

	date	favorite_count	retweet_count
10	Jan2020	29474	11454
9	Feb2020	36206	4116
8	Mar2020	6444	1471
7	Apr2020	6540	1430
6	May2020	236305	73725
5	Jun2020	20408	3334
4	Aug2020	12260	4821
3	Sep2020	4136	734
2	Oct2020	5368	1635
1	Nov2020	3	68
0	Dec2020	2391	855

Figure 5-7 Dataframe of most active time

```
#This is to get the January Text from the Adidas "User Timeline" and to conduct the analysis.
#This is used to downloaded dataset from the tweets collect functions
#Not the HOME TIMELINE OF ADIDAS!!
def getUserTimelineTweetPerformance(screen_name):
    #file = 'output/user_timeline_{}_between_1JanTo1Feb.json'.format(screen_name)
    file = 'output/user_timeline_{}_between_2020-2021.json'.format(screen_name)
    dic = {"date":[],"favorite_count":[], "retweet_count":[]}
    count=0
    with open(file) as f:
        for line in f:
            tweet = json.loads(line)
            favCount = tweet['favorite_count']
            retCount = tweet['retweet_count']
            d = tweet['created_at'].split(' ')
            dic['date'].append(d)
            dic['favorite_count'].append(favCount)
            dic['retweet_count'].append(retCount)
            count += 1

    user = api.get_user(screen_name)
    followersCount = user.followers_count
    #print("Followers Count : ",followersCount)
    dfDaily = pd.DataFrame(dic)
    print("Statuses Count : ",count)
    favouritesCount = dfDaily['favorite_count'].sum()
    print("Favourites Count : ",favouritesCount)
    retweetsCount = sum(dfDaily['retweet_count'])
    print("Retweets Count : ",retweetsCount)
    avgFavouritesCount = round(favouritesCount / count, 4)
    print("Average Favourite Count : ",avgFavouritesCount)
    avgRetweetsCount = round(retweetsCount / count, 4)
    print("Average Tweets Count : ",avgRetweetsCount)

    favoritPerUser = round(favouritesCount / followersCount, 4)
    retweetPerUser = round(retweetsCount / followersCount, 4)

    print("\n----- Statistic {} based on Favourties and Retweets -----".format(screen_name))
    print("Total {} followers".format(followersCount))
    print("Favourited {} times ({} per tweet, {} per user)".format(favouritesCount, avgFavouritesCount, favoritPerUser))
    print("Retweeted {} times ({} per tweet, {} per user)".format(retweetsCount, avgRetweetsCount, retweetPerUser))

    #return pd.DataFrame(dic)
getUserTimelineTweetPerformance("Adidas")
```

Figure 5-8 Content performance

Furthermore, we also generate some of the statistics based on the twitter data as shown in Figure 5-8. The statistics we generated are the favourite count, retweet count, statuses count, average tweet count, average favourite counts based on the one year data that we collected. The aim of generating the statistic is because we can clearly see the performance of the brands based on the favourite and retweet counts.

```
def getFollowers(screen_name):
    fname = "output/users/{}/followers.json".format(screen_name)
    print("Collecting followers for ", screen_name)
    with open(fname, 'w') as f:
        for followers in Cursor(api.followers_ids, screen_name=screen_name, count=500).pages(2):
            for page in paginate(followers, 100):
                users = api.lookup_users(user_ids=page)
                for user in users:
                    #print(user.screen_name)
                    f.write(json.dumps(user._json)+"\n")
    print("Followers collected")
getFollowers("Adidas")
```

```
Collecting followers for  Adidas
Followers collected
```

```
def getFriends(screen_name):
    fname = "output/users/{}/friends.json".format(screen_name)
    print("Collecting friends for ", screen_name)
    with open(fname, 'w') as f:
        for friends in Cursor(api.friends_ids, screen_name=screen_name, count=500).pages(2):
            for page in paginate(friends, 100):
                users = api.lookup_users(user_ids=page)
                for user in users:
                    f.write(json.dumps(user._json)+"\n")
    print("Friends collected")
getFriends("Adidas")
```

```
Collecting friends for  Adidas
Friends collected
```

Figure 5-9 Data collected for network graph

In Figure 5-9, as we mentioned in our metrics we retrieve the 1000 followers and friends from the brand we chose.

```
screen_name = "Adidas"
friends_file = 'output/users/{}/friends.json'.format(screen_name)
followers_file = 'output/users/{}/followers.json'.format(screen_name)
user_profile = 'output/users/{}/user_profile.json'.format(screen_name)

users = []
#with open(user_profile) as f:
#        profile = json.loads(line)
#        users.append(profile['screen_name'])
with open(friends_file) as f:
    for line in f:
        profile = json.loads(line)
        users.append(profile['screen_name'])
with open(followers_file) as f:
    for line in f:
        profile = json.loads(line)
        users.append(profile['screen_name'])
print(len(users))
```

```
1160
```

Figure 5-10 Inserting screen name into the array

After collecting the data, we have created an array to retrieve the screen name of the user and start collecting their followers and friends again.

```
from functools import partial

def get_friends_followers_ids(twitter_api, screen_name=None, user_id=None,
                              friends_limit=1000, followers_limit=1000):
    # Must have either screen_name or user_id (logical xor)
    assert (screen_name != None) != (user_id != None), \
    "Must have screen_name or user_id, but not both"

    #Here we start to collect the 1000 friends and followers from the screen name we given in the array
    #So we have set the limit 1000 for both the followers and friends retrieve
    get_friends_ids = partial(make_twitter_request, twitter_api.friends.ids,
                              count=1000)
    get_followers_ids = partial(make_twitter_request, twitter_api.followers.ids,
                              count=1000)
    #Here is creating the array to save the followers and friends ids based on the screen name
    friends_ids, followers_ids = [], []
    for twitter_api_func, limit, ids, label in [
            [get_friends_ids, friends_limit, friends_ids, "friends"],
            [get_followers_ids, followers_limit, followers_ids, "followers"]
            ]:
        if limit == 0: continue
        cursor = -1
        #Here is using a while loop to retrieve the ids based on the screen name in the array --
        while cursor != 0:
            if screen_name:
                response = twitter_api_func(screen_name=screen_name, cursor=cursor)
            else: # user_id
                response = twitter_api_func(user_id=user_id, cursor=cursor)
            if response is not None:
                ids += response['ids']
                cursor = response['next_cursor']
            print('Fetched {0} total {1} ids for {2}'.format(len(ids),
                    label, (user_id or screen_name), file=sys.stderr))
            if len(ids) >= limit or response is None:
                break
    return friends_ids[:friends_limit], followers_ids[:followers_limit]
```

Figure 5-11 Functions to retrieve followers and friends ids

In Figure 5-11, this is the function we reference from [4], [5]. In this function, it will compute the followers and friends ID from the screen name that we push into the array

```
#Here is starting the crawl the followers and friends based on the screen name
def crawl_followers(twitter_api, screen_name, limit=1000000, depth=2):

    # Resolve the ID for screen_name and start working with IDs for consistency
    seed_id = str(twitter_api.users.show(screen_name=screen_name)['id'])
    friends_ids, followers_ids = get_friends_followers_ids(twitter_api, user_id=seed_id,
                                 friends_limit=limit, followers_limit=limit)
    rp_friend = list(set(friends_ids) & set(followers_ids))
    top_five = get_user_profile(t, user_ids=rp_friend[:100])
    next_queue = top_five

    save_followers(seed_id, next_queue)
```

```
for name in users:
    crawl_followers(t, name, depth=1, limit=1000)
```

Figure 5-12 Starting to crawl the ids

As we mentioned previously, the friends and the followers of the brands will push into the users array. Hence, we used a for loop to crawl the followers for each of the screen names.

After finishing collecting the data, we have to do some simple cleaning for the dataset which drops the null columns. Besides, we also have to set the ids to integer or else it will show the ids as float numbers. The sample output as shown in Figure 5-13.

```
df = pd.read_csv("ReciprocalFriendAdidas.csv")
df.head()
```

	ID	ReciprocalFriend
0	1.771974e+07	9570452.0
1	NaN	NaN
2	4.634444e+09	133880286.0
3	NaN	NaN
4	4.634444e+09	22705078.0

```
df = df.dropna()
```

```
df = df.applymap(int)
```

```
df.head()
```

	ID	ReciprocalFriend
0	17719735	9570452
2	4634443760	133880286
4	4634443760	22705078
6	4634443760	757976000000000000
8	25254853	300114634

Figure 5-13 Simple data cleansing

```
# Create Graph from file
x_point = list(df[df.columns[0]].values)
y_point = list(df[df.columns[1]].values)
edges_list = []
for i in range(len(x_point)):
    edges_list.append((x_point[i], y_point[i]))
node_list = set(x_point+y_point)
RG = nx.Graph()
RG.add_nodes_from(node_list)
RG.add_edges_from(edges_list)
```

```
# Display some graph information such as number of nodes and edges
print ("Number of Nodes :", RG.number_of_nodes())
print ("Numebr of edges :", RG.number_of_edges())
```

```
Number of Nodes : 2392
Numebr of edges : 1963
```

Figure 5-14 Creating the nodes and edges

From the dataframe we extract out the data from the first and second column into a list respectively. After that, we insert the x point (source) and the y point (destination) into the edges list. Besides, we get the ids from both of the lists with a set() function.

```
# Calculate network density
# It shows how the connected nodes to each others.
print("Network Density", nx.density(RG))
```

Figure 5-15 Density of the graph

```
components = nx.connected_components(RG)
largestSubgraph = max(components, key=len)
print(largestSubgraph)
print(len(largestSubgraph))
```

```
#Need to rerun the component
minSubgraph = min(components, key=len)
print(minSubgraph)
print('len:',len(minSubgraph))
#print(min_subgraph)
```

Figure 5-16 Creating the subgraphs

In Figure 5-16, we have use the connected_components to help us compute the largest subgraph and smallest subgraph from the network graphs.

```
#First get the top ??? nodes by betweenness as a list
sortedBetweenness = sorted(betweenness_dict.items(), key=itemgetter(1), reverse=True)
topBetweenness = sortedBetweenness[:10]

#Then find and print their degree
for p in topBetweenness:
  print("User ID : ",p[0], "Degree: ", person_dict[p[0]],"Betweenness: ",p[1])
```

Figure 5-17 Determine the top betweenness from the graph

In Figure 5-17, we used a sorted() function to sort the top betweenness. We get the user id, degree and the betweenness to analyse these metrics.

Dashboard

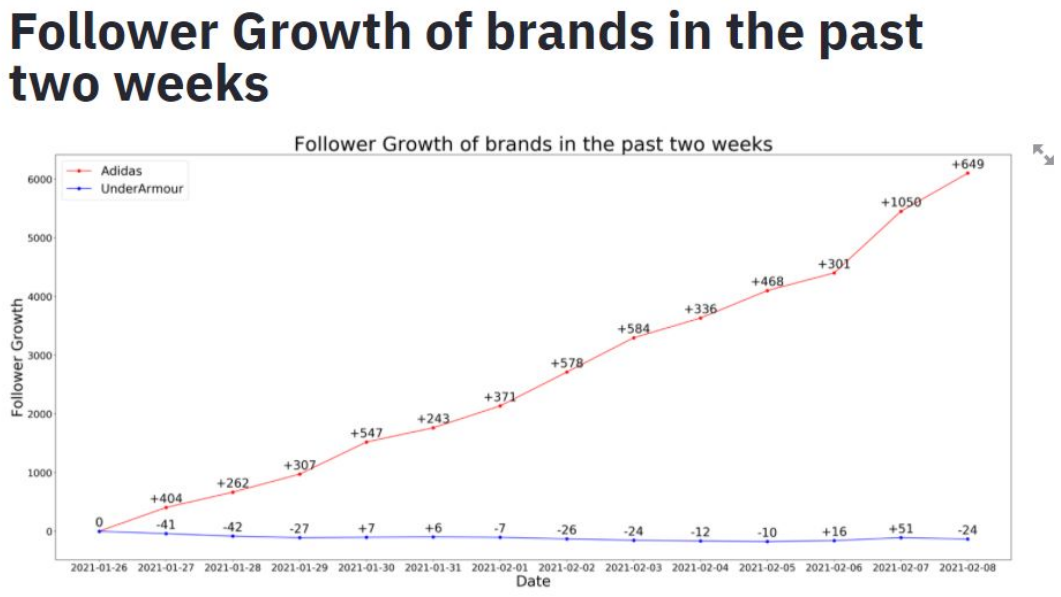

Figure 6-1 Follower growth

From Figure 6-1, we can observe that Adidas has a clear upward trend in these 2 weeks. The number of its followers increases between 200 to 1000 everyday. For the past two weeks, it has increased a total of 6100 number of followers. While UnderArmour has a relatively smooth trend in these 2 weeks as its number of followers decrease by a total of 133. In this case, we can see that Adidas is plainly better than UnderArmour.

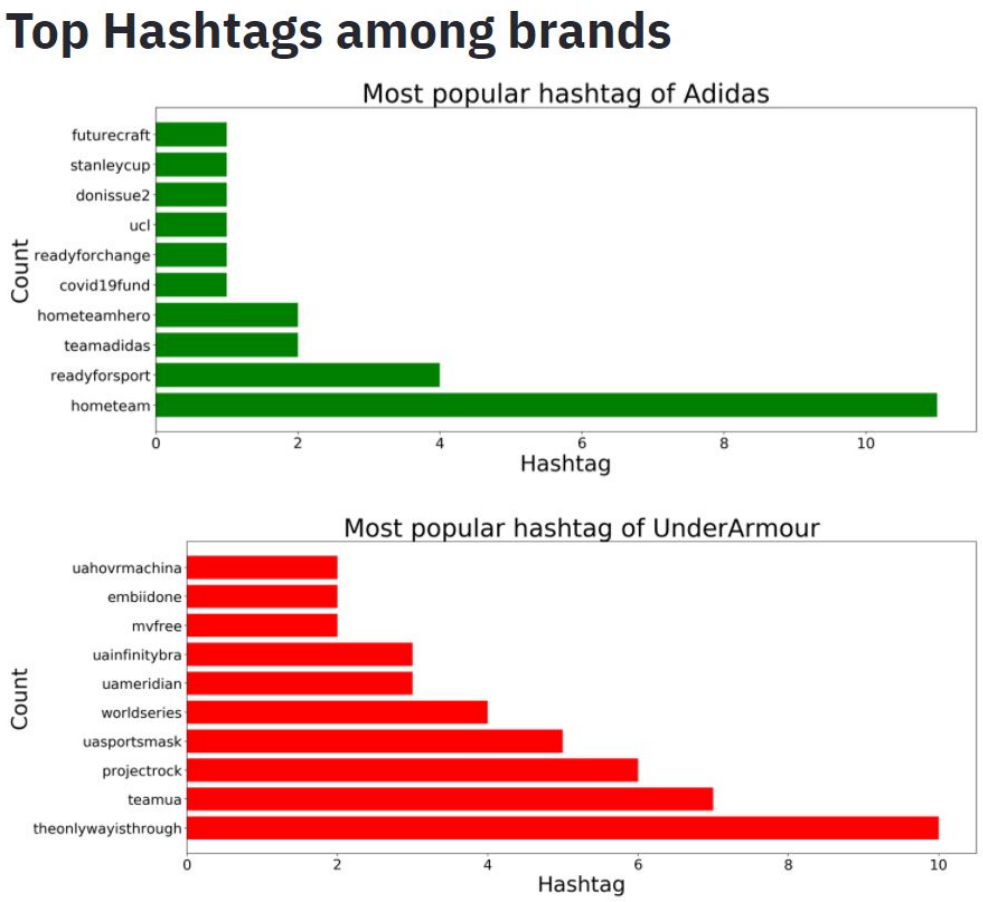

Figure 6-2 Top hashtags

Figure 6-2 shows the top 10 Hashtags among Adidas and UnderArmour. We can see that #hometeam and #theonlywayisthrough are the most used hashtags by Adidas and UnderArmour respectively which two of them had been used by the brands around 10 times. Most of the popular hashtags used are their events, products, sponsored athletes or sub-brands.

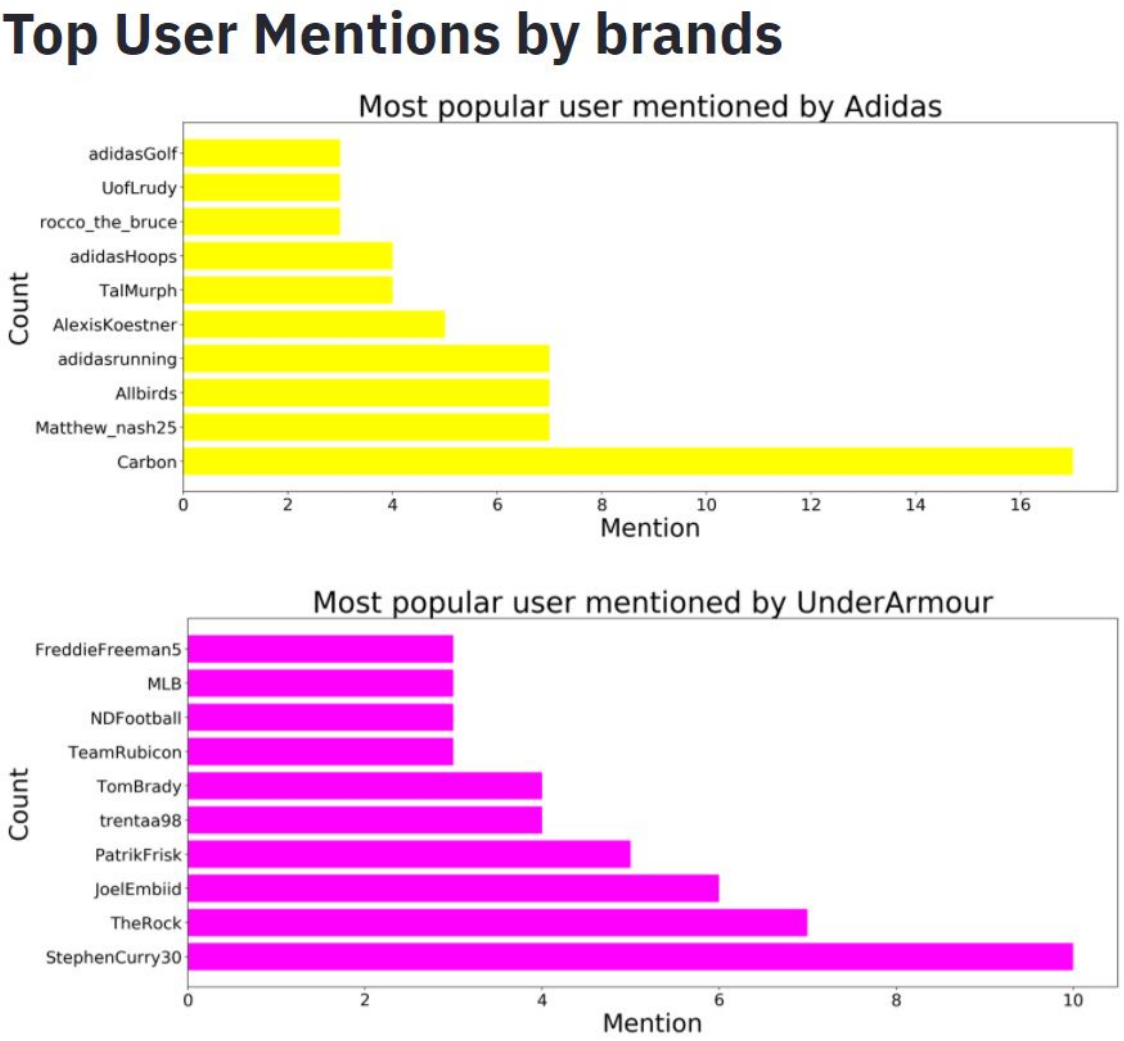

Figure 6-3 Top user mentions

Top user mentions by the two brands are very similar compared to the top hashtag metric as we can see that most of the users either are their sub-brands, journalists or sponsored athletes in Figure 6-3. For example, the most popular user mentioned by UnderArmour is StephenCurry30 who is a famous basketball player and Matthew_nash25 is a sports journalist of Adidas.

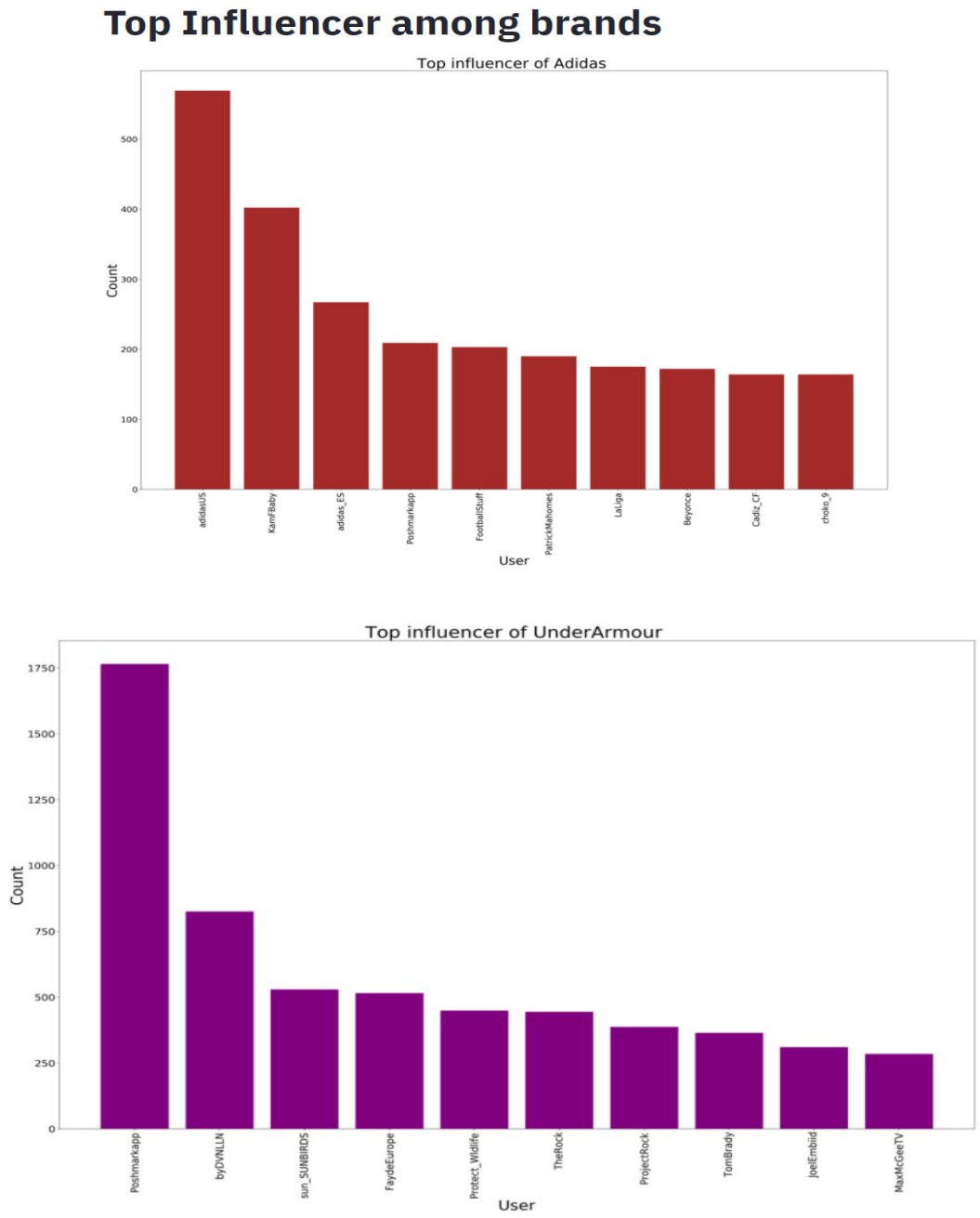

Figure 6-4 Top influencers

From Figure 6-4, we can see that adidasUS is the user who mentions Adidas the most with over 500 mentions. On the other hand, Poshmarkapp mentions UnderArmour around 1750 times. For the overall, we can clearly see that UnderArmour definitely has more users mentioning it than Adidas because UnderArmour has a larger y-axis scale compared to Adidas, which is 0 to 1750. While Adidas has a scale of 0 to 500.

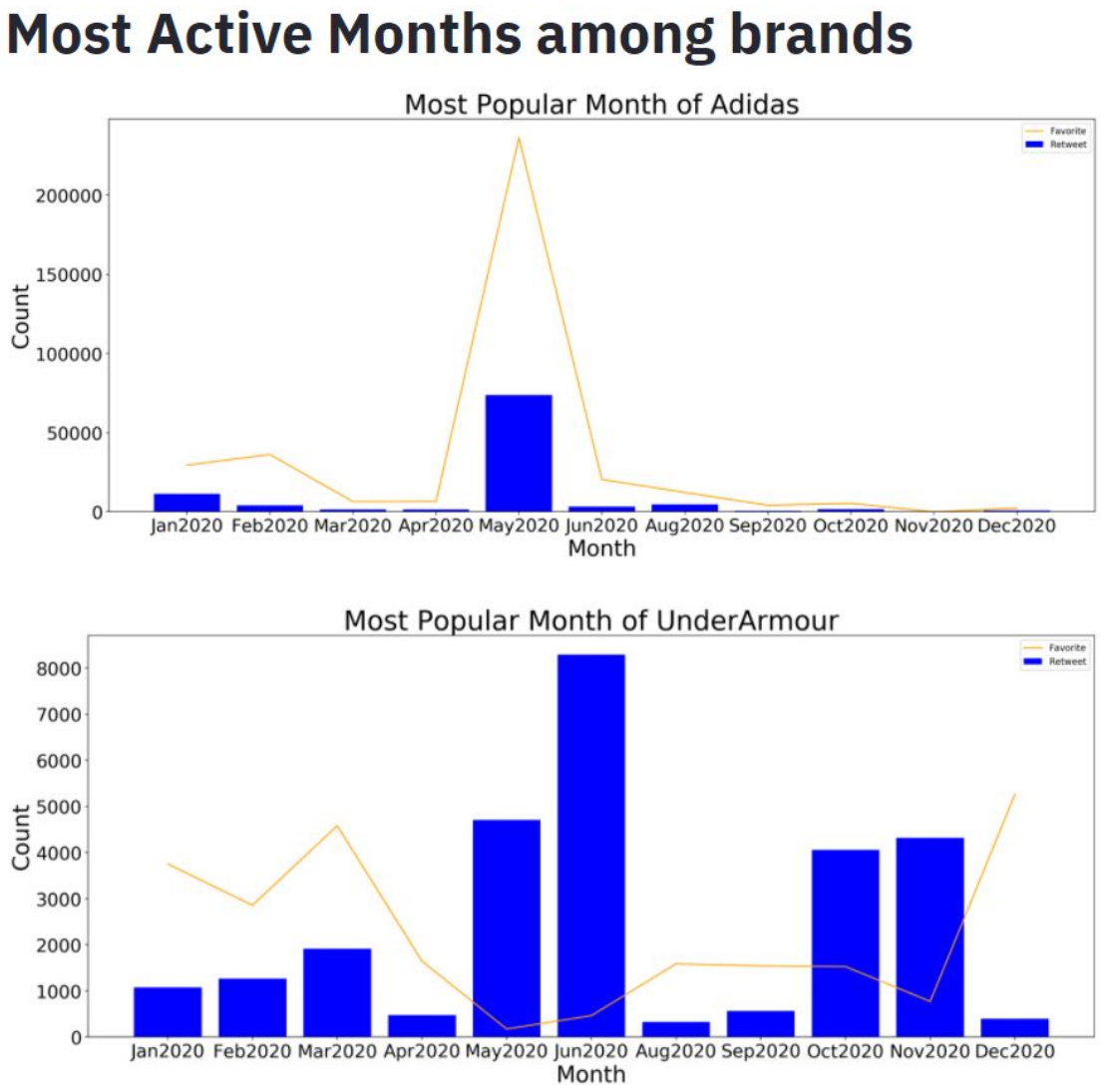

Figure 6-5 Most active time

From Figure 6-5, we can observe that Adidas has the most favorites and retweets in May 2020 as it has over 200,000 favorites and almost 100,000 retweets. This is mainly because Adidas shared a video of Nike which is against racial discrimination in 30 May 2020. This tweet has about 228,000 favorites and 78,000 retweets. As for UnderArmour, it has its highest favorite in December 2020 because one of its sponsored athletes, Anthony Joshua has successfully defended his champion. This tweet has about 2,900 favorites. On Jun 2020, UnderArmour retweeted its sponsored athlete tweet which is also against racial discrimination. This tweet has around 8,600 retweets. Overall, Adidas is more active because it has a larger y-axis scale compared to UnderArmour.

Comparison of content performance among brands

	Adidas	UnderArmour	Winner
Followers Count	3858431.0	950814.0	Adidas
Statuses Count	199.0	129.0	Adidas
Favourites Count	359535.0	24172.0	Adidas
Retweets Count	103643.0	27411.0	Adidas
Average Favourite Count	1806.7085	187.3798	Adidas
Average Retweets Count	520.8191	137.7437	Adidas
Favorite Per User	0.0932	0.0254	Adidas
Retweet Per User	0.0269	0.0288	UnderArmour

Figure 6-6 Content Performance

Figure 6-7 Radar chart content performance

From the table and radar chart above, we can clearly see that Adidas dominates UnderArmour in almost every aspect except for retweet per user.

Network Analysis

Adidas

Network Statistics of Adidas:

	Adidas
Nodes	2392.0
Edges	1963.0
Network Density	0.0006864510028549088
Length of largest subgraph	41.0
Length of smallest subgraph	2.0

Figure 7-1 Network Statistics of Adidas

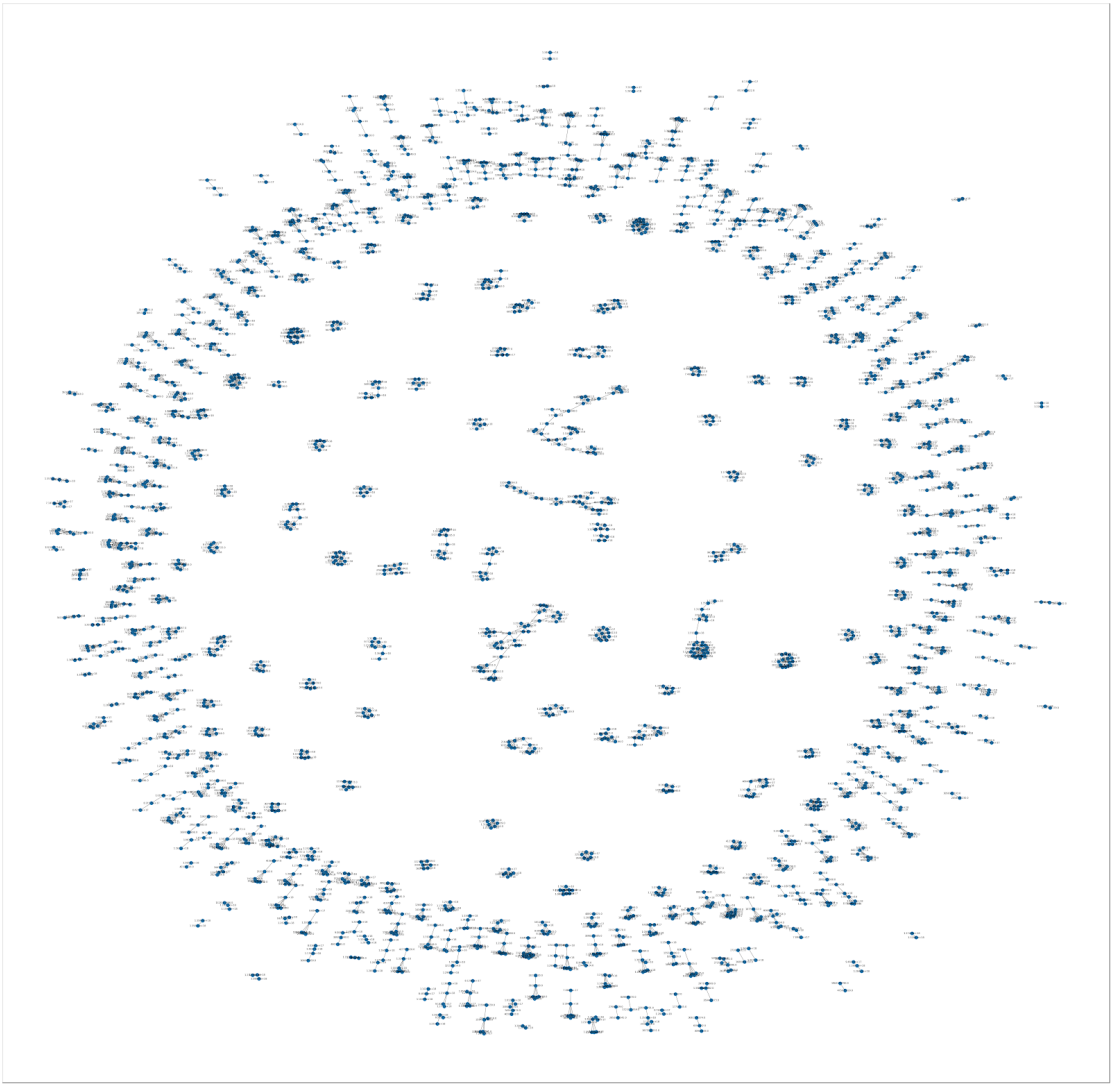

Figure 7-2 Network graph of Adidas

From the table of Figure 7-1, we can observe that in the network of Adidas, it has 2392 number of nodes and 1963 number of edges. The network density is 0.00068, which means that all of the nodes are not that related to each other. Other than that, the largest subgraph has a length of 41 while the length of the smallest subgraph of this network is just 2.

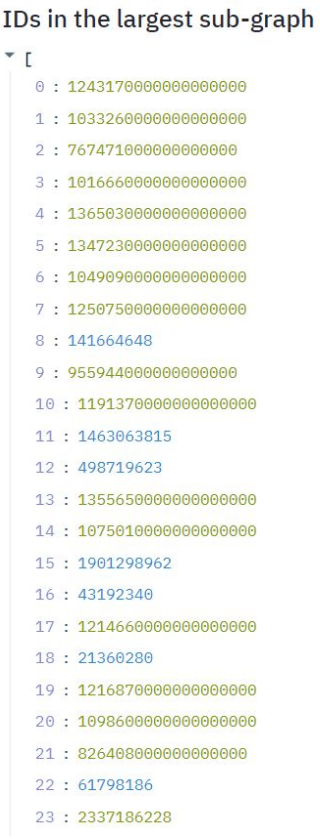

Figure 7-3 nodes in the largest subgraph in network of Adidas

In Figure 7-3, it shows 23 nodes out of 41 total nodes from the largest subgraph.

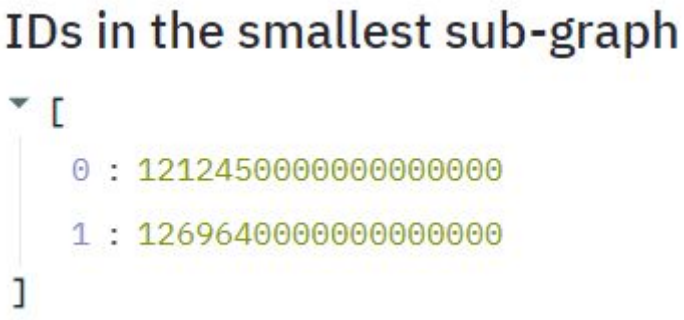

Figure 7-4 nodes in the smallest subgraph in network of Adidas

Figure 7-4 shows all the nodes from the smallest subgraph in the network of Adidas, with only 2 of the nodes connected to each other.

Top 10 nodes by betweenness centrality:

	Name	Degree	Betweenness
0	141664648	4	0.0001837434311723356
1	712019258	5	0.00017499374397365294
2	1901298962	3	0.00011374593358287441
3	1365470000000000000	20	0.0001130459586069798
4	1365030000000000000	5	0.00010499624638419177
5	1184790000000000000	5	0.00010342130268842888
6	1186830000000000000	5	9.992142780895583e-05
7	1339500000000000000	3	8.784685947477378e-05
8	1170400000000000000	2	8.644690952298456e-05
9	300114634	5	8.434698459530073e-05

Figure 7-5 Top 10 nodes by betweenness centrality

In the table of Figure 7-5, we can see that the node with the highest betweenness is ID 141664648, which has 4 degrees and a betweenness of 0.00018. This node is the most central node in connecting any pairs of nodes in this network. The betweenness is computed by the number of shortest paths between other nodes passing through this node.

Top 10 nodes by eigen centrality:

	Name	Degree	Eigen
0	1365470000000000000	20	0.7055335449138733
1	1311080000000000000	2	0.16799213217357817
2	1270780000000000000	1	0.15750001162341318
3	4310770282	1	0.15750001162341318
4	2243142523	1	0.15750001162341318
5	88876311	1	0.15750001162341318
6	999069000000000000	1	0.15750001162341318
7	1338740000000000000	1	0.15750001162341318
8	306514241	1	0.15750001162341318
9	1184920000000000000	1	0.15750001162341318

Figure 7-6 Top 10 nodes by eigenvector centrality

From Figure 7-6, we can see that ID 1365470000000000000 is the node with the highest eigenvector centrality with a score of 0.70553. It means that this node is connected to many nodes who themselves also have high scores as well.

Top 10 nodes by closeness centrality:

	Name	Degree	Closeness
0	1365470000000000000	20	0.007090543024714774
1	1365580000000000000	15	0.006273525721455458
2	141664648	4	0.005974786401386151
3	712019258	5	0.005921912185444681
4	1365600000000000000	14	0.005855290673358427
5	1311080000000000000	2	0.005543515455686096
6	1365760000000000000	13	0.005437055625261397
7	1901298962	3	0.005069515734509461
8	1365030000000000000	5	0.005031399074851495
9	300114634	5	0.004791981229043943

Figure 7-7 Top 10 nodes by closeness centrality

In the table of Figure 7-7, we can see that ID 1365470000000000000 is also the highest closeness centrality node with a score of 0.007. The score means that this node is more connected. In addition, it can reach other nodes quickly as they have smaller average shortest paths.

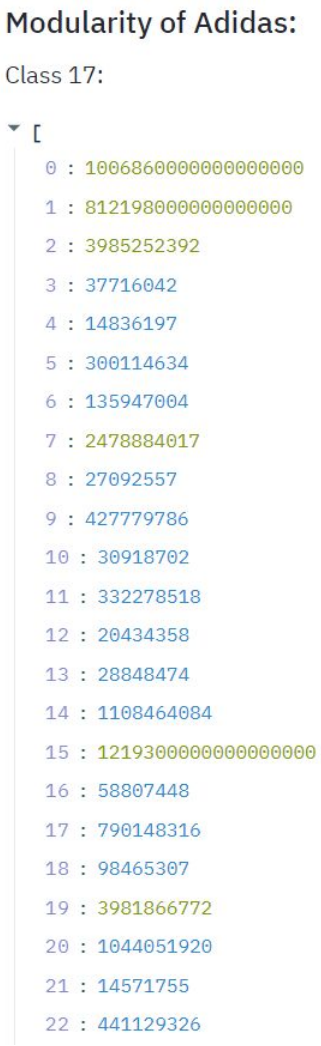

Figure 7-8 A class of Adidas's modular communities

Figure 7-8 shows one of the classes of Adidas' modular communities. As Adidas' network should be far from random, modularity defines a distance which the more distant Adidas' network is from a randomly generated network, the more structural it is. Additionally, modularity maximization will try to maximize this distance.

UnderArmour

Network Statistics of UnderArmour:

	UnderArmour
Nodes	2433.0
Edges	2111.0
Network Density	0.0007135305124710666
Length of largest subgraph	464.0
Length of smallest subgraph	2.0

Figure 7-9 Network Statistics of Adidas

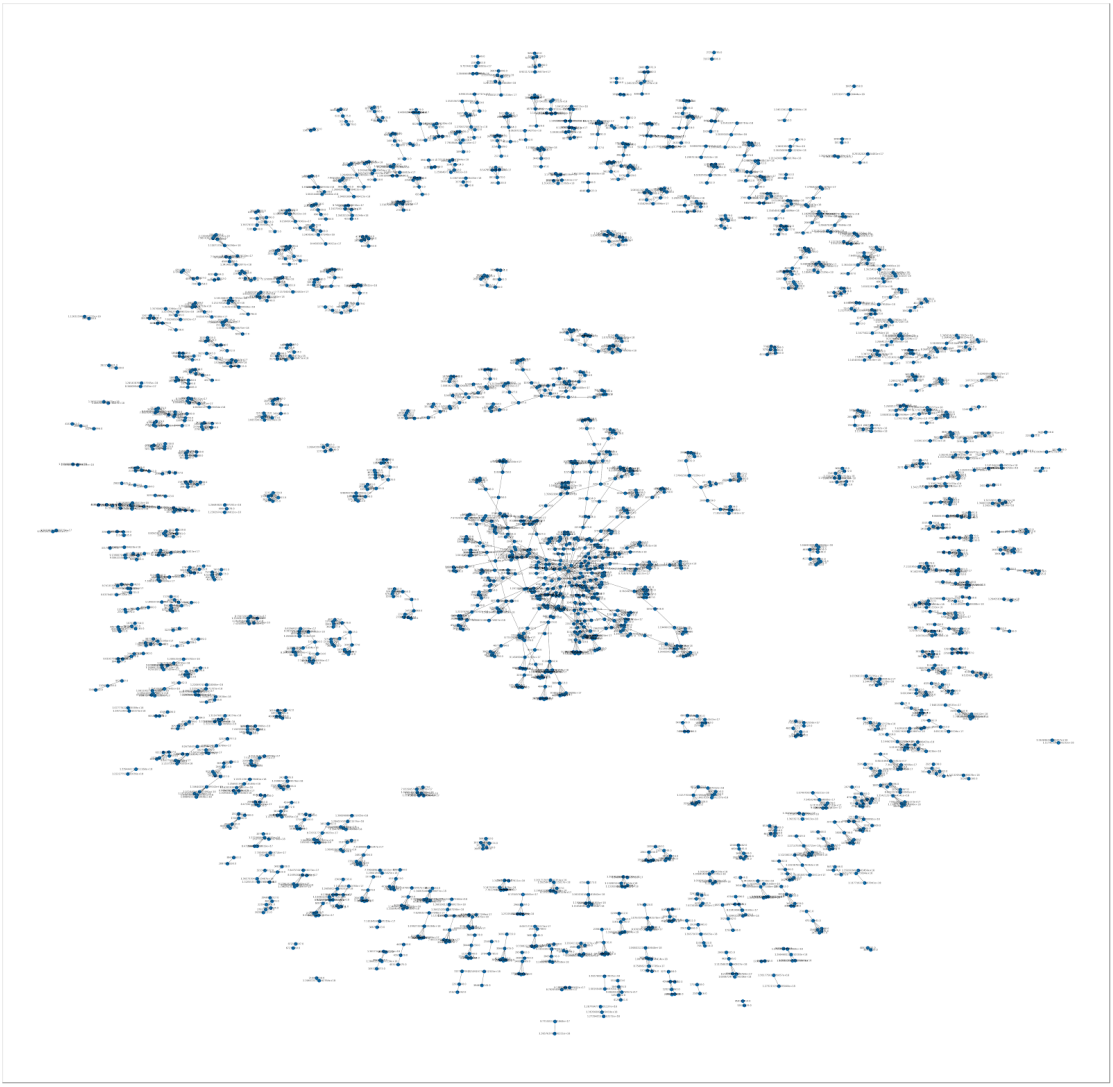

Figure 7-10 Network graph of UnderArmour

In the network graph of UnderArmour, it has 2433 number of nodes along with 2111 number of edges. For the network density, it has 0.00071 which is slightly higher than Adidas's network. In this network, it has a huge subgraph in which 464 nodes are linked together. While the smallest subgraph has only 2 nodes connected.

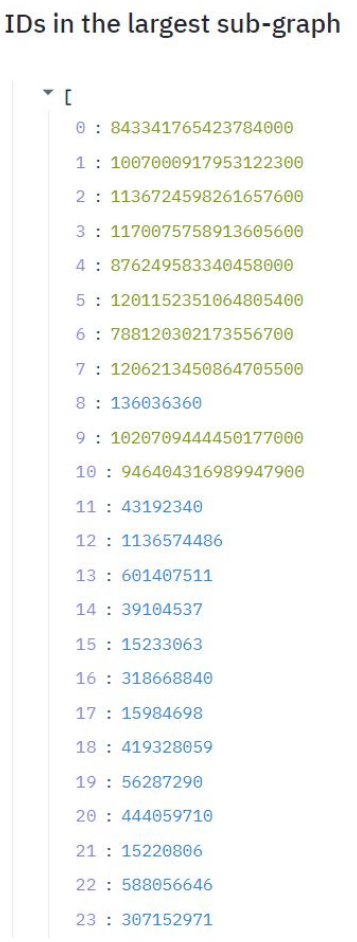

Figure 7-11 nodes in the largest subgraph in network of UnderArmour

In the largest subgraph of the UnderArmour network, it has a total of 464 number of nodes, which is a huge one.

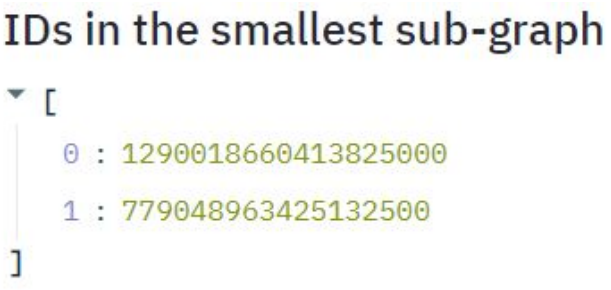

Figure 7-12 nodes in the smallest subgraph in network of UnderArmour

While in the smallest graph of the UnderArmour network, there are only 2 existing nodes.

Top 10 nodes by betweenness centrality:

	Name	Degree	Betweenness
0	23114836	35	0.02853963954002652
1	312745095	8	0.017841460977103388
2	22451734	5	0.013361928319358665
3	319805618	5	0.006338427439433632
4	219823828	5	0.006308094640137984
5	1170075758913605632	5	0.005452799909069259
6	2508496374	5	0.005426848693102562
7	602188858	3	0.005118237026131763
8	22686121	6	0.00496302104757987
9	505934719	6	0.004320563337591202

Figure 7-13 Top 10 nodes by betweenness centrality

We can observe that ID 23114836 is the most central node in the network of UnderArmour as it has 35 degrees along with a betweenness of 0.28539 from Figure 7-13. It is the most central node in this network.

Top 10 nodes by eigen centrality:

	Name	Degree	Eigen
0	23114836	35	0.6543237813829597
1	859106874	5	0.13654867883027028
2	818014362	5	0.1329943653047356
3	412323221	5	0.1298191220068994
4	523917886	5	0.1298191220068994
5	190762445	5	0.12981912200689938
6	170164405	5	0.127722182479273
7	1545931003	5	0.12747762753202993
8	1585374968	5	0.12456760774073786
9	930989796	5	0.12425166466015473

Figure 7-14 Top 10 nodes by eigenvector centrality

From the figure above, we can observe that ID 23114836 has a highest score of 0.65432 in eigenvector centrality. It is connected to other high eigenvector centrality score nodes.

Top 10 nodes by closeness centrality:

	Name	Degree	Closeness
0	23114836	35	0.04767179449773704
1	22451734	5	0.044138782186437554
2	312745095	8	0.04199387709686317
3	319805618	5	0.03965143860832919
4	2508496374	5	0.0396157968657599
5	859106874	5	0.03929788142055987
6	818014362	5	0.0392628721720783
7	412323221	5	0.0392628721720783
8	190762445	5	0.0392628721720783
9	523917886	5	0.0392628721720783

Figure 7-15 Top 10 nodes by closeness centrality

ID 23114836 has the highest centrality score from the table of Figure 7-15. It can reach other nodes quicker because they have smaller average shortest paths.

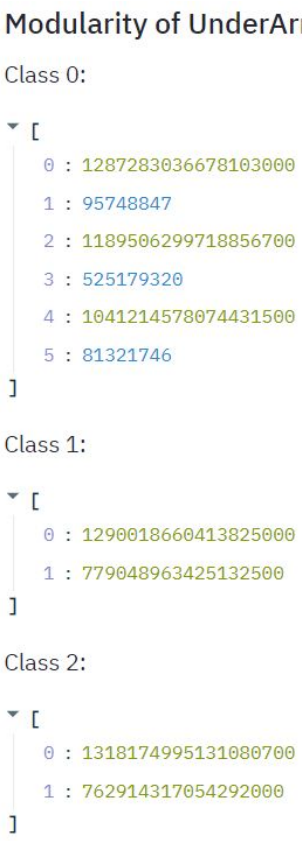

Figure 7-16 A class of UnderArmour's modular communities

Figure 7-16 shows one of the classes of UnderArmour's modular communities. Same as Adidas' case, UnderArmour's network should be far from random, modularity will define a distance which the more distant UnderArmour's network is from a randomly generated network, the more structural the network is. Furthermore, modularity maximization will try to maximize this distance as far as possible.

Conclusion

Which brand amongst our selected ones succeeds? From our analysis, we found the most suitable answer is Adidas. As in the comparison between Adidas and UnderArmour, Adidas wins 4 out of 6 metrics. First of all, Adidas definitely wins in the follower growth metric. Secondly, the 2 brands draw for top hashtags and top users mention. Moreover, we give UnderArmour a win for the top influencer metric. For the last two metrics, which are most active time and content performance, Adidas unquestionably outperforms UnderArmour. Therefore, the final score for the comparison between Adidas and UnderArmour is 5:3. Hence, we can say that Adidas is the more successful one in using their social media channel absolutely.

Reference

[1] 6 ways social media managers can deliver a great Twitter experience

[2] Hootsuite: Twitter Marketing: The Complete Guide for Business

[3] 10 Examples of Successful Twitter Hashtag Campaigns [Updated- 2020]

[4] Using Python And NetworkX To Build A Twitter Follower Recommendation Engine

[5] Bhushan-Jagtap-2013/Community-Detection-in-Graphs: Built friendship graph from Twitter data and performed community detection.

[6] Why Twitter is the Ideal Platform for Engagement | Convince and Convert