Exploring Hacker News Posts

Introduction

The aim of this project is to study Hacker News posts, specifically, we are going to analyse the *Ask* and *Show* posts:

- **Ask posts**: users submit questions to the Hacker News community.
- **Show posts**: users share personal projects, products, or something interesting related to computing or entrepreneurship.

In this analysis both types of posts are being compared in order to answer these questions:

- Which type of post (*Ask* or *Show*) receive more comments on average?
- Do posts created at a specific time frame receive more comments on average?

The data set was extracted from (Kaggle), however it has been reduced from almost 300,000 rows to approximately 20,000 rows by removing all submissions that didn't receive any comments and then randomly sampling from the remaining submissions (DataSet).

We start by reading the dataset as a list of lists, and dispalying the five rows:

In [1]:
```
#We import the module for reading
from csv import*
open_file=open('hacker_news.csv')
read_file=reader(open_file)
hn=list(read_file) #We call our data set 'hn' (Hacker News)

#We print the first five rows to see how the data set looks  like:
print(hn[:5])
```

```
[['id', 'title', 'url', 'num_points', 'num_comments', 'author', 'created_at'], ['12224879', 'Interactive Dynamic Video', 'http://www.interactivedynamicvideo.com/', '386', '52', 'ne0phyte', '8/4/2016 11:52'], ['10975351', 'How to Use Open Source and Shut the Fuck Up at the Same Time', 'http://hueniverse.com/2016/01/26/how-to-use-open-source-and-shut-the-fuck-up-at-the-same-time/', '39', '10', 'josep2', '1/26/2016 19:30'], ['11964716', "Florida DJs May Face Felony for April Fools' Water Joke", 'http://www.thewire.com/entertainment/2013/04/florida-djs-april-fools-water-joke/63798/', '2', '1', 'vezycash', '6/23/2016 22:20'], ['11919867', 'Technology ventures: From Idea to Enterprise', 'https://www.amazon.com/Technology-Ventures-Enterprise-Thomas-Byers/dp/0073523429', '3', '1', 'hswarna', '6/17/2016 0:01']]
```

As wee see, the first row of our data sets corresponds to dataset headers. Hence, we are going to separate them:

In [2]:
```
headers=hn[0]
print(headers)
#We remove the first row of hn
hn=hn[1:]
print(hn[:5]) #We check headers have been removed
```

```
['id', 'title', 'url', 'num_points', 'num_comments', 'author', 'created_at']
[['12224879', 'Interactive Dynamic Video', 'http://www.interactivedynamicvideo.com/', '386', '52', 'ne0phyte', '8/4/2016 11:52'], ['10975351', 'How to Use Open Source and Shut the Fuck Up at the Same Time', 'http://hueniverse.com/2016/01/26/how-to-use-open-source-and-shut-the-fuck-up-at-the-same-time/', '39', '10', 'josep2', '1/26/2016 19:30'], ['11964716', "Florida DJs May Face Felony for April Fools' Water Joke", 'http://www.thewire.com/entertainment/2013/04/florida-djs-april-fools-water-joke/63798/', '2', '1', 'vezycash', '6/23/2016 22:20'], ['11919867', 'Technology ventures: From Idea to Enterprise', 'https://www.amazon.com/Technology-Ventures-Enterprise-Thomas-Byers/dp/0073523429', '3', '1', 'hswarna', '6/17/2016 0:01'], ['10301696', 'Note by Note: The Making of Steinway L1037 (2007)', 'http://www.nytimes.com/2007/11/07/movies/07stein.html?_r=0', '8', '2', 'walterbell', '9/30/2015 4:12']]
```

Data Analysis

We will start by filtering the data. We only want to analyse *Ask* and *Show* post, so let's separate these types of posts from the rest:

In [3]:
```
#We start by creating three empty lists:
ask_posts=[]
show_posts=[]
other_posts=[]

#We loop over the hn list to separate the posts:
for row in hn:
    title=row[1] #The title of the post (Ask, Show,...) corresponds with the second column
    #We may use the startswith method to filter posts:
    title=str(title) #We need to work with strings for lowercases
    if ((title.lower()).startswith('ask hn')):
        ask_posts.append(row)
    elif ((title.lower()).startswith('show hn')):
        show_posts.append(row)
    else:
        other_posts.append(row)

#Let's check the length of each list
print('There are '+ str(len(ask_posts))+ ' ask posts.')
print('There are '+ str(len(show_posts))+ ' show posts.')
print('There are '+ str(len(other_posts))+ ' posts which are not aks or show post.')
```

```
There are 1744 ask posts.
There are 1162 show posts.
There are 17194 posts which are not aks or show post.
```

Now we are going to find the average Ask and Show comments to see which one recieves more comments on average.

First we will determine the total number of comments of each section, and then divide them by the total number of posts:

In [4]:
```
#Ask comments:
#First we calculate the total number of comments:
total_ask_comments=0
for row in ask_posts:
    comments=int(row[4])
    total_ask_comments=total_ask_comments+comments

#Now we calculate the average
average_ask=total_ask_comments/len(ask_posts)
print('The average of Ask comments is of: %.4f ' %(average_ask))

#Show comments:
#We repeat the same process
total_show_comments=0
for row in show_posts:
    comments=int(row[4])
    total_show_comments += comments

average_show=total_show_comments/len(show_posts)
print('The average of Show comments is of: %.4f ' %(average_show))
```

```
The average of Ask comments is of: 14.0384
The average of Show comments is of: 10.3167
```

As we may see, for each Ask post there are 14 comments, while on Show post ther are only, on average, 10 comments per post. Hence, we are going to analyse if Ask posts created at a specific time frame receive more comments on average.

First of all, we are going to determine the number of Ask posts created in each hour of the day, along with the number of comments received.

For carrying out this task, the `datetime` module is going to be used.

In [5]:
```
#We import this module as dt to not have ambiguous syntax
import datetime as dt

#We create an empty list, that will be a list of lists:
result_list=[]
for row in ask_posts:
    creation=row[6] #The column that indicates the creation date of the post
    comments=int(row[4]) #The column  that indicates the number of comments of the post
    result_list.append([creation, comments])

#Now lets create two dictionaries:
counts_by_hour={}
comments_by_hour={}
for row in result_list:
    date=row[0] #We extract the dates
    comments=int(row[1]) #We extract the comments
    dates=dt.datetime.strptime(date, '%m/%d/%Y %H:%M') #We convert the dates (string) to a datetime object
    hour=dates.strftime('%H') #We select only hour and convert datatime object to string
    #With conditionals we create frequency tables
    if hour not in counts_by_hour:
        counts_by_hour[hour] = 1
        comments_by_hour[hour] = comments
    else:
        counts_by_hour[hour] += 1
        comments_by_hour[hour] += comments

#We check that both dictionaries work properly
print(counts_by_hour)
print(comments_by_hour)
```

```
{'09': 45, '13': 85, '10': 59, '14': 107, '16': 108, '23': 68, '12': 73, '17': 100, '15': 116, '21': 109, '20': 80, '02': 58, '18': 109, '03': 54, '05': 46, '19': 110, '01': 60, '22': 71, '08': 48, '04': 47, '00': 55, '06': 44, '07': 34, '11': 58}
{'09': 251, '13': 1253, '10': 793, '14': 1416, '16': 1814, '23': 543, '12': 687, '17': 1146, '15': 4477, '21': 1745, '20': 1722, '02': 1381, '18': 1439, '03': 421, '05': 464, '19': 1188, '01': 683, '22': 479, '08': 492, '04': 337, '00': 447, '06': 397, '07': 267, '11': 641}
```

Now let's calculate the average number of comments for posts created during each hour of the day:

In [6]:
```
#We create an empty list
avg_by_hour=[]
for i in comments_by_hour:
    avg_by_hour.append([i, comments_by_hour[i]/counts_by_hour[i]])
#We divide comments / post to calculate the average comments for each post.
print(avg_by_hour)
```

```
[['09', 5.577777777777775], ['13', 14.741176470588234], ['10', 13.440677966101696], ['14', 13.233644859813085], ['16', 16.796296296296298], ['23', 7.985294117647059], ['12', 9.41095890410959], ['17', 11.46], ['15', 38.59482758620069], ['21', 16.009174311926607], ['20', 21.525], ['02', 23.810344827586206], ['18', 13.20183486238532], ['03', 7.796296296296297], ['05', 10.086956521739313], ['19', 10.8], ['01', 11.383333333333333], ['22', 6.746478873239437], ['08', 10.25], ['04', 7.170212765957447], ['00', 8.127272727272727], ['06', 9.022727272727273], ['07', 7.852941176470588], ['11', 11.051724137931034]]
```

We have the results, but not in a nice format for analysing them. So, as we want to know if there's an especific time frame were post receiver more comments, we are going to sort the list and print the 5 highest values in a format that's easier to read

In [13]:
```
#We create a list equal to avg_by_hour but wth swapped columns:
swap_avg_by_hour=[]
for i in avg_by_hour:
    swap_avg_by_hour.append([i[1], i[0]])
#Let's print this list
print(swap_avg_by_hour)
#We sort the list in descending order with the function `sorted()`
sorted_swap=sorted(swap_avg_by_hour, reverse=True)
#We add reverse = True, so that the highest value in the first column
#appears first in the list.
print('\n')
#Let's print the Top 5 Hours for Ask Posts Comments:
print(sorted_swap[:5])
```

```
[[5.577777777777775, '09'], [14.741176470588234, '13'], [13.440677966101696, '10'], [13.233644859813085, '14'], [16.796296296296298, '16'], [7.985294117647059, '23'], [9.41095890410959, '12'], [11.46, '17'], [38.59482758620069, '15'], [16.009174311926607, '21'], [21.525, '20'], [23.810344827586206, '02'], [13.20183486238532, '18'], [7.796296296296297, '03'], [10.086956521739313, '05'], [10.8, '19'], [11.383333333333333, '01'], [6.746478873239437, '22'], [10.25, '08'], [7.170212765957447, '04'], [8.127272727272727, '00'], [9.022727272727273, '06'], [7.852941176470588, '07'], [11.051724137931034, '11']]


[[38.5948275862069, '15'], [23.810344827586206, '02'], [21.525, '20'], [16.796296296296298, '16'], [16.009174311926607, '21']]
```

Now, we are going to change the format, and `print()` the hour and average like: *15:00: 38.59 average comments per post.*

In [16]:
```
#We use the str format method:
print("Top 5 Hours for 'Ask HN' Comments")
for avg, hr in sorted_swap[:5]:
    print("{}: {:.2f} average comments per post".format(dt.datetime.strptime(hr, "%H").strftime("%H:%M"),avg))
#Truco -> Si tenemos una fecha o como en este caso hora de tipo string, primero tenemos que pasarla a datetime object
#Para ello usamos strptime y tenemos que indicar en el formato en el que esta con los porcentajes.
#Posteriormente, una vez en datetime object, podemos cambiarle el formato con strftime (Por ej pasar de 12:00 a 12 am)
#Cuando hacemos esto, strftime hace que el datetime object vuelva a ser un string.
```

```
Top 5 Hours for 'Ask HN' Comments
15:00: 38.59 average comments per post
02:00: 23.81 average comments per post
20:00: 21.52 average comments per post
16:00: 16.80 average comments per post
21:00: 16.01 average comments per post
```

This times refer to the time zone in Eastern US. However, we are going to use GMT (Greenwich Mean Time). The Eastern Time Zone (ET) is an area 5 hours behind Greenwich Mean Time.

We can see that the best hour to submit and Ask post, is at 10:00 (GMT), with an average of 38.59 comments per post.

A point worth noting is that there's a **61.84% increase** in the number of comments between the highest and second highest average number of comments.

Conclusion

In this project, we analyzed ask posts and show posts to determine which type of post and time receive the most comments on average. We found that Ask post receive more comments per post on average. Furthermore, based on our analysis, to maximize the amount of comments a post receives, we'd recommend the post be categorized as ask post and created at 10:00 (GMT).

However, it should be noted that the data set we analyzed excluded posts without any comments. Given that, it's more accurate to say that of the posts that received comments, Ask posts received more comments on average and Ask posts created at 10:00 (GMT) received the most comments on average.