Movie’s Gross Earnings Correlation to Twitter Factors

Author: Jonathan Bracci

SI 330 Data Manipulation

April 18, 2017

INTRODUCTION:

Initially my project was to use the New York Times API and pull the top financial stories and then pull key words from these stories and run them through quandl, which was a financial data base that I found. I could pull the New York Times API successfully and gather the key words using NLTK, but the problem that I kept running into was that I needed the company names or the stock market symbol. I found this part very difficult because many times the company was not mentioned enough or when it was pulled out there were too many additional other frequent words that were selected as well causing me many errors. This is a project that I will continue to work on in the future, but for the given time constraint I decided to switch my project.

Given that I wasted many hours on trying to get my initial project to work I then decided to do something that we covered more closely in class. I decided to use the IMDB movie website for top 50 grossing movies in the United States. What caught my interest about the IMDB website was that I wanted to see additional correlations rather than actors, how we did earlier in the year. The main variable that I wanted to examine was how gross earnings for the top movies related to other independent variables. The variables that I wanted to see if there were correlations with the year the movie was produced, imdb\_rating, votes the movie received on IMDB. This would all be able to be done from the IMDB data, but I also wanted to run it against an additional data set with a more real time response. For this I used the Twitter API to look at tweets within a set period that contained the movie title in them and look at a correlation of gross earnings to amount of favorites the tweets got and the average polarity of the tweets.

DATA SOURCES:

1. Parsed IMDB html & Used OMDB API
   1. Location
      1. Html: <http://www.imdb.com/search/title?genres=action&sort=boxoffice_gross_us,desc>
      2. OMDB API:

**http://www.omdbapi.com/?i='** + movie[**'IMDB\_ID'**]

* 1. Size
     1. The size of these data bases are all of the information which can be accessed on the website, which could not be found online.
  2. Format
     1. HTML , JSON Dictionary, Csv, Txt
  3. Import Methods
     1. urllib.request, urlopen, BeautifulSoup, json, os

1. Twitter API
   1. Location
      1. Twitter AP - Tweepy:

auth = tweepy.OAuthHandler(consumer\_key,consumer\_secret)  
auth.set\_access\_token(access\_token,access\_token\_secret)api = tweepy.API(auth)

* 1. Size
     1. Access to all of twitters 317 million users, with 100 million active users per day, and 500 million tweets per day.
  2. Format
     1. Csv, Txt
  3. Import Methods
     1. tweepy, csv, TextBlob, os

DATA PROCESSING STEPS:

First I ran the IMDB\_Data.py file, the first data processing step is the function fetch\_top50\_grossing\_movies() it uses urlopen to access the html page that is specified, passing in that url. This returns an html format that needs to be read in through the .read() method and then decoded by utf-8 to make it readable for the human user. Now the data is named and written into an top50.html to be parsed through in the next function. Now the extract\_movie\_info() function is called and immediately a new csv file called movie\_info.csv is opened and strings are written into the headers for information to be inserted into. The with open method is now used on the html file created above. Then the BeautifulSoup module is used to read in the top50.html to create a parse tree to more easily extract targeted data, such as imdb\_title. A loop is used to iterate through each movie row and the imdb\_id finds all the ‘a’ tags with an ‘href’ attribute and is then split by ‘/’, to extract the proper data to then be appended to a dictionary with their rank and title as well to be referred to later by the OMDB API to extract each movies metadata. This data which is stored in a dictionary is used to lookup values from their index which are then output into a top50.txt.

Now that we have all that data it is time to pull all the metadata using the OMDB API. The top50.txt file is read in and put in a loop to get each movies metadata as JSON and appended to a list called out, giving us a list of dictionaries, most importantly giving us keys that we can call to get specific values for. By doing this we are then able to iterate through the list of dictionaries and for each movie we are able to do an API call by calling the ‘<http://www.omdbapi.com/?i>= + movie[‘IMDB\_ID’]’. This is taking this in as a string and then the API call is being imported into the out list to give all the metadata which is put out in a JSON format. As stated above now this data is again imported to an additional file called imdb\_meta.txt

Then the extract\_more() function is called to read through the metadata, which is a JSON object and to output into an additional csv file called Final\_Info.csv to get a better representation of the data to further manipulate later in the project. Once again a loop is used to iterate through the data in order to read one line at a time and a json.loads function is used with in the loop to make sure it is not a JSON object and we can call key value pairs to output the selected values into the Final\_Info.csv

The next step is to open the file called twitter\_puller.py. Now that I could pull the IMDB data I then analyzed that and used the tweepy module to access the Twitter API. To access this I had to get an access\_token, access\_token\_secret, consumer\_key, and consumer\_secret. Each user needs these to connect and pull data from the Twitter API. I then authenticated these variables and set api=tweepy.API(auth), so that when I call the variable that I defined as api it will use the tweepy module with my credentials to call the twitter API.

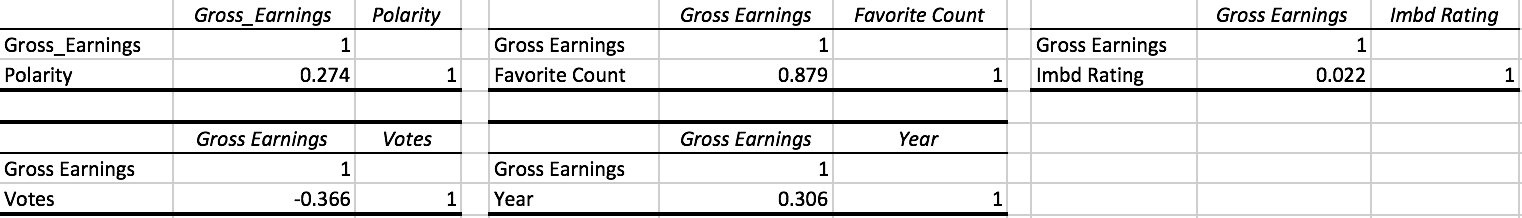
Then I defined 5 functions in my main definition in addition to setting variables for key word searches and file names to then be called when each individual function was called. I will walk through one of them because the rest of them are the same just with a few different parameters. The first function that is called is fetch\_Star\_Wars(filename). Since this is defined in the main function it then looks it up and sees that the filename is called Star\_Wars\_Results.csv and the key word being parsed through to look for tweets that contain that word is Star Wars.

The Star\_Wars\_Results.csv is then opened in write mode and I first write the headers into the csv as strings to then later called them to append all the twitter data to this csv file. The ids variable is set equal to set(), I used a set because it is similar to a tuple, but did not want duplicate ids in order to count the number of different users for the tweets. Then a loop is put into place to iterate through tweepy. Cursor(api.search). This calls the Twitter API and then additional parameters such as the keyword, since, until, count, and language are specified to extract only certain tweets in a given period of time. The Twitter API has restrictions on how many tweets can be pulled at a given time for the REST API, which is what limited the number of tweets I was able to pull. But if one uses the streaming you can run this for a set period to gather more information, but you must keep it running the whole time to grab the data. In the loop these objects are returned as tweepy objects, which allows for us to call built in tweepy functions, such as: .created\_at, .favorite\_count, .tweet. Then these objects were appended to the csv file for each tweet. Finally, the tweet ids were added to the id set and then counted the number of unique ids that showed up in the pulled data. Then this process was repeated for the additional top grossing movies and all put into separate csv files.

RESULTS:

By using the OMDB API I could find additional results such as number of votes, rating, and year the movie titles were produced. But this it was unknown when this data was last updated and how it would compare to the data about these titles today. Therefore, the data alone could give us correlations, but one could not be sure if the correlations still stood today. That is where the twitter data could come into play because I was able to pull tweets with in the last week and compare that to the top grossing movies, regardless of when they were produced.

Since my data did not contain the same headers I was unable to create a relational database with themselves. Instead I imported all the data from the individual csv files into one csv file called combine.csv. From there I used excel and could extract separate columns from each separate worksheet. For example, I created a new worksheet with the headers: Title, Gross Earnings, Favorite Count, Polarity, Retweet Count, imdb\_rating, Votes, Year, and Rank. The imdb info could have been a relational database before, but not the twitter information, allowing me to put all this information into one excel worksheet to look for correlations.



*Created in Excel*

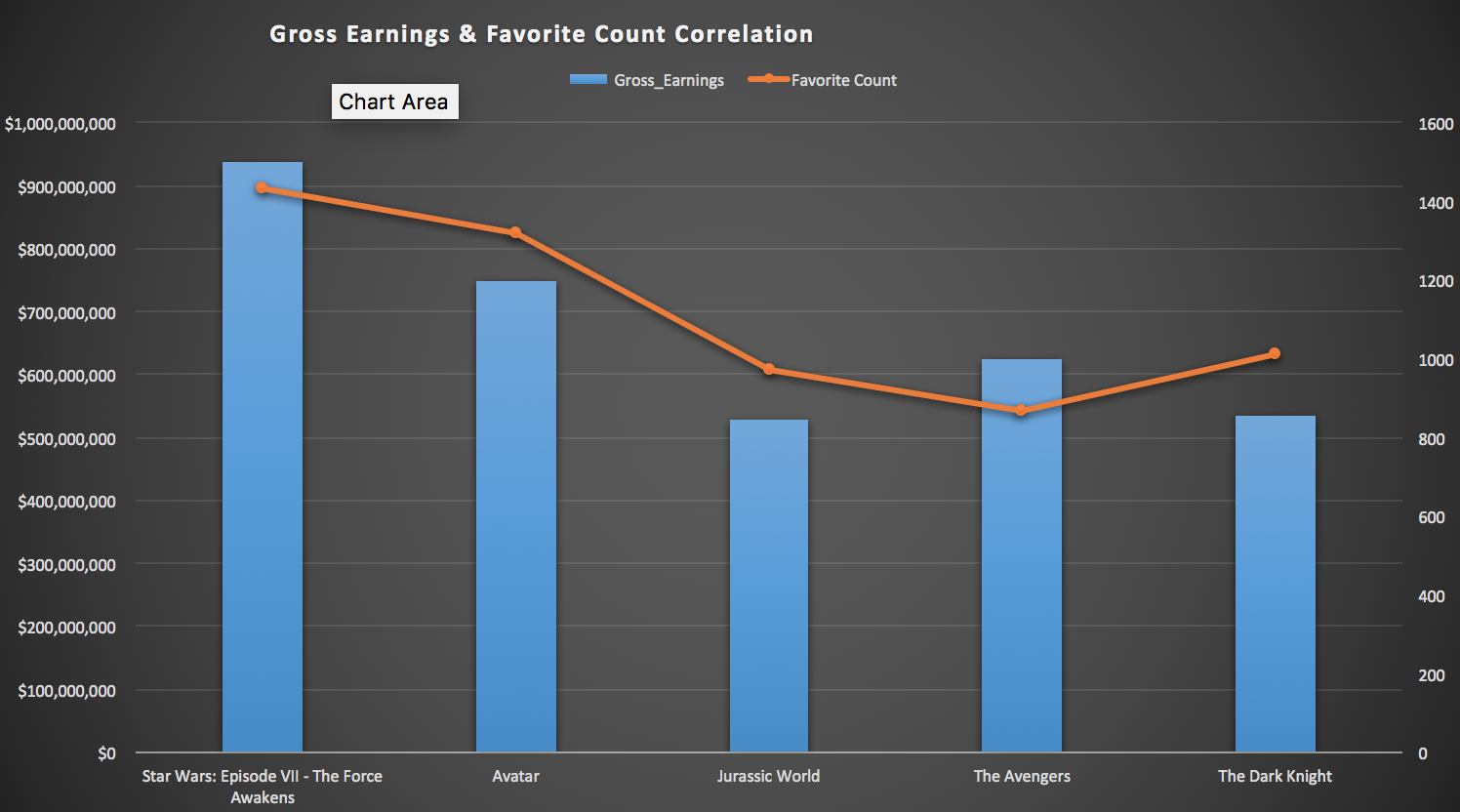
*Author: Jonathan Bracci*

*Correlation of independent variable to gross earnings (Described Below)*

I could see how each variable correlated with gross earnings by making 5 separate tables and then using the CORREL function, which is a built in excel function to calculate the correlation between the two variables.

Out of the 5 factors that I tested against Gross Earnings the most significant correlation that I found was between the Gross Earnings for movie titles & Tweet Favorite Count. The correlation that I calculated was a positive 0.879, with 1 meaning that the two variables are perfectly correlated. When two variables have a strong positive correlation, this means generally as one variable increases the other one will increase as well. A large correlation can range from 0.5 – 1.0, medium from 0.3 – 0.5, and small from 0 – 0.3. Adversely goes the same for a negative correlation, except in this case as one variable was increasing the other would more likely be decreasing, giving them a negative correlation. As seen in the charts below the Polarity, imdb\_rating, and Year of production all had a small positive correlation. While, number of votes has a medium negative correlation and favorite count has a strong positive correlation to gross earnings.

From these results, I wanted to see how they looked graphed against each other on the y –axis’ with the movie title on the x-axis. Then I created a combo chart for each putting gross earnings on the left y-axis for each and the additional variable on the right y-axis to keep consistency to see how each compared with each other and against gross earnings. As seen below you can see how as gross earnings for a title increases the number of favorites for a tweet also increases.



*Created in Excel*

*Author: Jonathan Bracci*

*The rest of the visualizations can be found in combine.csv*

*Correlation of Favorite count on Gross Earnings (Described Above)*

Discussion & Conclusion:

Although I was only able to find one relationship that showed a strong correlation. I could see how relationship formed in some cases and were non-existent in others. Through this I could see that as one item is more popular, such as higher grossing movie sales, other factors that relate directly to it are more likely to have a positive correlation, such as tweet favorites. In addition to this a positive correlation with gross earnings had a positive correlation, likely because of inflation not just because of popularity.

If I were to do further analysis on this project one this I would try to further look at the year they were produced against the relative value of their gross earnings, which would mitigate the effects of inflation. In addition to this I would using the tweepy stream module and pull data over a longer period. For example, pull data for two days straight at the beginning of each month and compare the trends over the course of a year. My sample size was large enough to make conclusions, but with a larger sample size I would be able to make stronger connections and be more confident as well.

This has also brought me back to modify my initial project idea and look at into trends to see how stock prices are affected based on the frequency they are tweeted about and the polarity of each tweet.

Overall this project helped me better understand pulling data from an API and parsing through HTML documents by using the BeautifulSoup module. Prior to this project, I had previously pulled data from API’s, but I had never pulled such large amounts of information and exported them into files. This project really helped me to better understand how to pull data and export the data into a csv file. I find it easier to further manipulate and organize the data by using excel because I have more experience, but as shown in class python is also very powerful in creating visuals and it is something I want to work on moving forward. This project proved to be very beneficial and I was able to understand data structures better. Moving forward I want to continue to experiment with API’s and HTML files because the only way to get better at them is to continue to practice.

References

Tweepy Documentation & Help :

<http://docs.tweepy.org/en/v3.5.0/>

<https://pythonhosted.org/tweepy/api.html>

Omdb Documentation & Help :

<http://omdbapi.com/>

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

<https://docs.python.org/3/library/csv.html>

SI 330 HW5