Final Project

Social Media Analytics

Chih-Hsiang Lai
Chih-Hua Lin
Vivek Bhatia
Yu-Jui Chen

Table of Contents

TEAM LIST

Name	Email	Program	Role of each member
Chih-Hsiang Lai	chlai2@illinois.edu	MSTM-ADV	Project Lead, Project coding, Data Visualization, Data Modeling, Prediction
Vivek Bhatia	vcb2@illinois.edu	MSTM-ADV	Data Collection, Documentation, Presentation
Yu-Jui Chen	yujuijc2@illinois.edu	MSTM-ADV	Data Collection, Data Modeling, Prediction, Data Visualization, Business Analysis
Chih-Hua Lin	chihhua2@illinois.edu	MSTM-ADV	Data Collection, Data Modeling, Prediction, Data Visualization, Documentation

INTRODUCTION

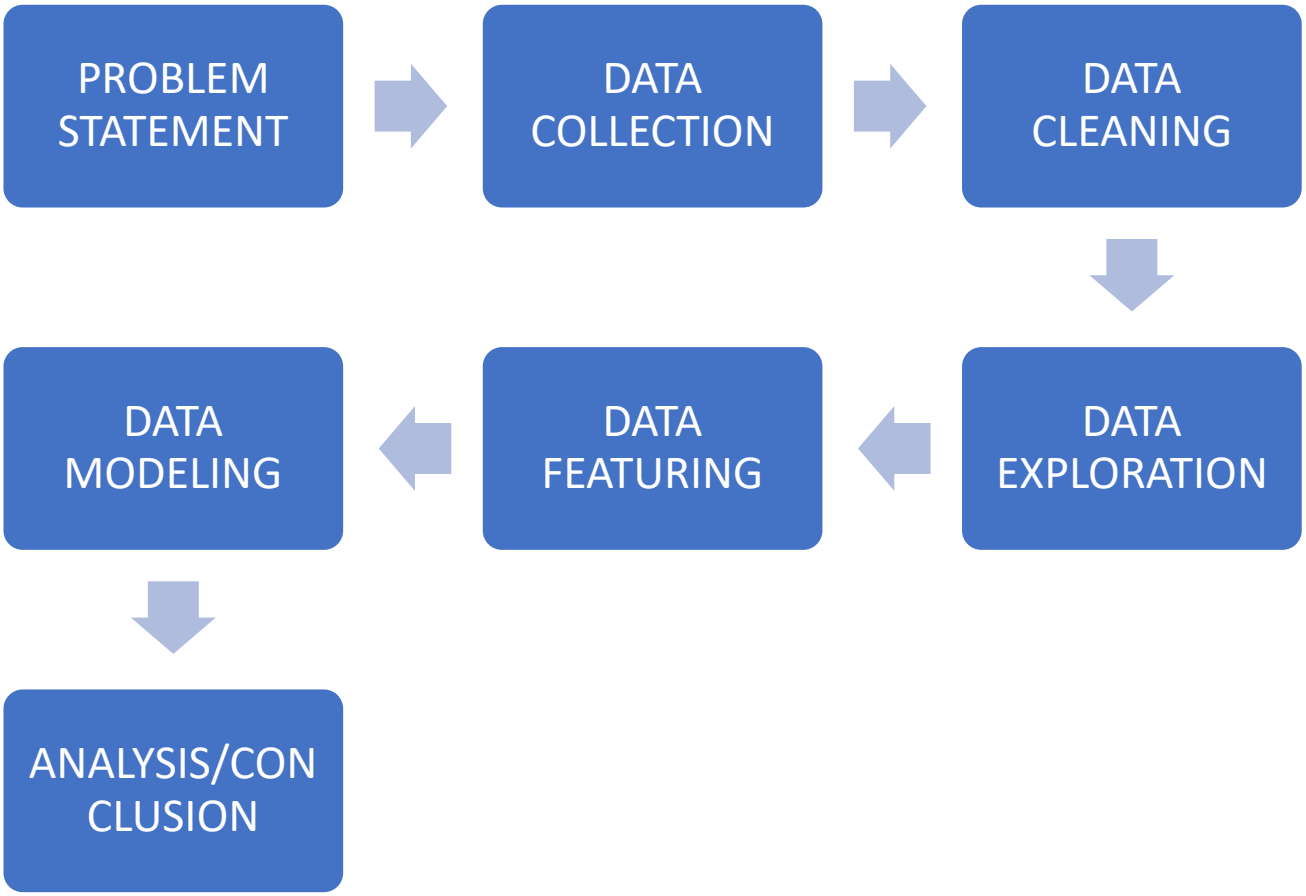
BACKGROUND

OBJECTIVE

With millions of dollars being invested in movies every year, a lot is at stake on how your favorite film fairs at the box office. With so many factors involved in making a blockbuster possible, there is a plethora of things that could decide its fate. We try to solve this dilemma by considering these various factors that vary for each movie and in turn affect the revenue generated by it. Using various modelling techniques, we built a prediction model to reach the goal of successfully predicting the revenue of a movie. Machine learning and Linear regression form the basis of our analysis. Using 80% of the data to train the model we first form the fodder for the machine learning algorithm. We then use the 20% data to test our results. This combination of analyses and inference based on machine learning provides an optimized output.

PROCESS AND DELIVERABLES

PROCESS MAP



PROBLEM STATEMENT

To predict the weekly revenue of a movie with respect to Runtime, Rating, Votes, Genre, Director, Lead Actor, Number of Tweets, Number of Retweets and the number of Positive, Negative Tweets.

DATA COLLECTION

1. **MOVIE LIST**

The source of the movie lists is the online platform Kaggle. The data set contains 1,000 most popular movies on IMDB from year 2006 till year 2016. The data points included are: Title, Genre, Description, Director, Actors, Year, Runtime, Rating, Votes, Revenue and Metascore.

Variable	Description	Data type
Rank	Movie rank order	Numeric
Title	The title of the film	String
Genre	A comma-separated list of genres used to classify the film	String
Description	Brief one-sentence movie summary	String
Director	The name of the film's director	String

Variable	Description	Data type
Actors	A comma-separated list of the main stars of the film	String
Year	The year that the film released as an integer.	Numeric
Runtime (Minutes)	The duration of the film in minutes.	Numeric
Rating	User rating for the movie 0-10	Numeric
Votes	Number of votes	Numeric
Revenue (Millions)	Movie revenue in millions	Numeric
Metascore	An aggregated average of critic scores. Values are between 0 and 100. Higher scores represent positive reviews.	Numeric

2. WEEKLY GROSS

Collected weekly domestic box-office dataset from Mojo website based on the list of 1,000 most popular movies on IMDB from year 2006 till year 2016.

Variable	Description	Data type
Movie Name	The title of the film	String
Date	First date of the week number of the ongoing film in the theaters	Date/Time
Weekly Gross	Weekly gross of the film	Numeric
Parentage_Change	Weekly gross parentage change of the film	Numeric
Avg	Average weekly gross of the film from all the domestic theaters	Numeric
Gross_to_Date	Weekly cumulative gross of the film	Numeric
Week_Number	The week number of the ongoing film	Numeric

1000 Movies IMDB from 2006 to 2016 [resource: Kaggle.com]

	Unnamed: 0	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore	Un
0	1	Guardians of the Galaxy	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13	76.0	
1	2	Prometheus	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7	485820	126.46	65.0	
2	3	Split	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2016	117	7.3	157606	138.12	62.0	
3	4	Sing	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2016	108	7.2	60545	270.32	59.0	
4	5	Suicide Squad	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2016	123	6.2	393727	325.02	40.0	

Weekly Revenue for 1000 Movies [resource: mojo.com]

	Movie_Name	Date	Weekly_Gross	Percentage_Change	Avg	Gross_to_Date	Week_Number
0	Guardians of the Galaxy	Aug 1?	$134,390,839	-	$32,939	$134,390,839	1
1	Guardians of the Galaxy	Aug 8?4	$63,154,982	-53.00%	$15,449	$197,545,821	2
2	Guardians of the Galaxy	Aug 15?1	$36,708,036	-41.90%	$9,929	$234,253,857	3
3	Guardians of the Galaxy	Aug 22?8	$24,043,411	-34.50%	$7,132	$258,297,268	4
4	Guardians of the Galaxy	Aug 29ÐSep 4	$26,110,586	8.60%	$7,542	$284,407,854	5
5	Guardians of the Galaxy	Sep 5?1	$13,476,978	-48.40%	$4,184	$297,884,832	6
6	Guardians of the Galaxy	Sep 12?8	$10,604,199	-21.30%	$3,416	$308,489,031	7
7	Guardians of the Galaxy	Sep 19?5	$6,914,244	-34.80%	$2,429	$315,403,275	8
8	Guardians of the Galaxy	Sep 26ÐOct 2	$4,923,129	-28.80%	$2,009	$320,326,404	9
9	Guardians of the Galaxy	Oct 3?	$3,943,500	-19.90%	$2,082	$324,269,904	10
10	Guardians of the Galaxy	Oct 10?6	$2,562,827	-35.00%	$1,985	$326,832,731	11
11	Guardians of the Galaxy	Oct 17?3	$1,262,858	-50.70%	$1,730	$328,095,589	12
12	Guardians of the Galaxy	Oct 24?0	$730,948	-42.10%	$1,818	$328,826,537	13

3. TWEETS

Collecting tweets on twitter for all the 1,000 most popular movies on IMDB from year 2006 till year 2016. The main approach is by using python script to scratch data for each movie, which included the following variables.

Variable	Description	Data type
Movie Name	The title of the film	String
TweetsNum	Total number of the tweet per day	Numeric
Total_Likes	Total number of the like per day	Numeric
Total_RTs	Total number of the retweet per day	Numeric
Sum_SA_Likes	Weekly (sum of SA * sum of Like)	Numeric
Sum_SA_RTs	Weekly (sum of SA * sum of RTs)	Numeric
Sum_SA_is_retweet	Weekly (sum of SA * sum of retweet)	Numeric
Num_is_RTs		Numeric

Variable	Description	Data type
Positive	Weekly sentiment analysis result of positive	Float
Negative	Weekly sentiment analysis result of negative	Float

Tweet data set look

A	C	X	Y	Z	AA	AB	AC	AD	AE	AF
	Movie_Name	TweetsNum	Total_Likes	Total_RTs	Sum_SA_Likes	Sum_SA_RTs	Sum_SA_is_retweet	Num_is_RTs	Positive	Negative
0	Star Wars: The Force Awal	4203	4673	1923	1023	344	0	0	0.35474661	0.07328099
1	Star Wars: The Force Awal	9858	12725	5824	3862	1371	0	0	0.331000203	0.151957801
2	Star Wars: The Force Awal	12615	32645	14998	25395	11091	0	0	0.375267539	0.091082045
3	Star Wars: The Force Awal	2784	2401	935	546	149	0	0	0.345905172	0.061422414
4	Star Wars: The Force Awal	359	681	283	31	0	0	0	0.342618384	0.077994429

Twitter web-crawler Explanation

In this project, we aimed to utilize twitter data to predict how social buzz will affect movie revenue. To analyze the weekly gross of a movie, we need the data for movies weeks by weeks. Therefore, we built new columns on the weekly revenue dataset, *since_date* and *until_date*, in the purpose of scraping data from twitter. We also created 8 weeks of columns before the movie release, trying to analyze how twitter discussion of a movie will affect it.

We supposed to use twitter's API that was introduced in the class. However, we discovered that we were unable to access the data one week before today due to the API limitation. To solve this problem, we wrote a web-crawler which could extract all the twitter data we needed.

In our web-crawler code **[Exhibit A]**, we firstly used a selenium package to build a browser-based driver that will scraping all the tweets' ids we wanted to search. The webdirver will access to the url and scrolling the page directly to attain the ids and generate a json file of ids. **[Exhibit N]**.

After having the twitter id, the code will get the whole status from the website, transferring them in to a json file. Then we can extract the columns and content we need from that json file. Because we wrote it as a function, we could only input the original dataset from weekly revenue to get the final data (shown as the above figure.)

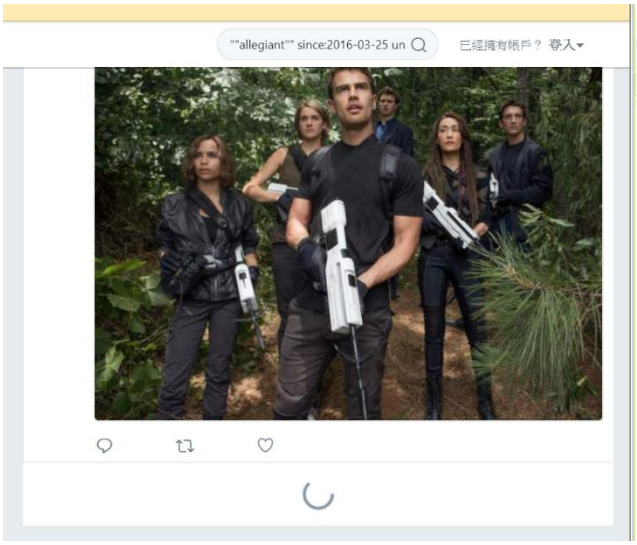
Exhibit A

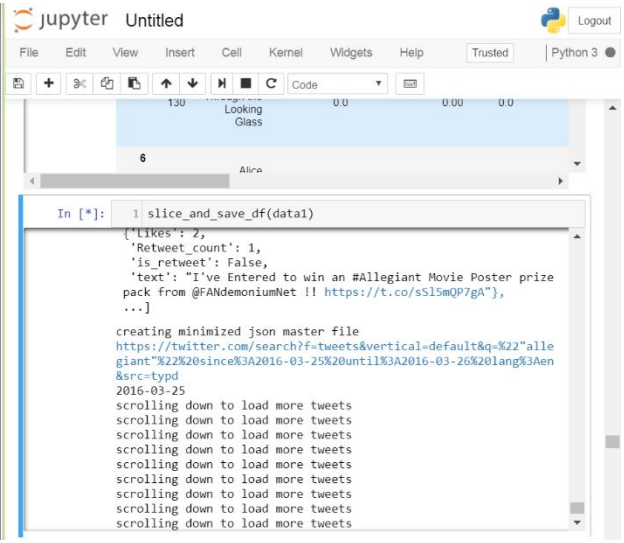
```python
def Get_twitter_data(keywards,start,end):

    ################################## Set Keywords, since, until, output file names ##################################
    # CHANGE THIS TO THE USER YOU WANT
    # edit these three variables
    #keywards = 'sphygmomanometer'
    user = keywards
    #start = datetime.datetime(2016, 12, 7)  # year, month, day
    #end = datetime.datetime(2016, 12, 8)  # year, month, day
    start_date=str(start).split(' ')[0]
    user = user.lower()

    output_file = '{}.json'.format(user.split("\"")[1]+start_date)
    output_file_short = '{}_short.json'.format(user.split("\"")[1]+start_date)
    compression = zipfile.ZIP_DEFLATED
    ################################## Set Keywords, since, until, output file names ##################################

    ################################################ Getting Tweets IDs ################################################
    # only edit these if you're having problems
    delay = 1  # time to wait on each page load before reading the page
    driver = webdriver.Chrome("C:\Driver\chromedriver.exe")  # options are Chrome() Firefox() Safari()

    # don't mess with this stuff
    twitter_ids_filename = 'all_ids_{}.json'.format(user.split("\"")[1]+'_'+start_date)
    days = (end - start).days + 1
    id_selector = '.time a.tweet-timestamp'
    tweet_selector = 'li.js-stream-item'
    user = user.lower()
    ids = []

    #set id files with driver
    for day in range(days):
        d1 = format_day(increment_day(start, 0))
        d2 = format_day(increment_day(start, 1))
        url = form_url(user, d1, d2)
        print(url)
        print(d1)
        driver.get(url)
        sleep(delay)

        try:
            found_tweets = driver.find_elements_by_css_selector(tweet_selector)
            increment = 10

            while len(found_tweets) >= increment:
                print('scrolling down to load more tweets')
                driver.execute_script('window.scrollTo(0, document.body.scrollHeight);')
                sleep(delay)
                found_tweets = driver.find_elements_by_css_selector(tweet_selector)
                increment += 10

            print('{} tweets found, {} total'.format(len(found_tweets), len(ids)))

            for tweet in found_tweets:
                try:
                    id = tweet.find_element_by_css_selector(id_selector).get_attribute('href').split('/')[-1]
                    ids.append(id)
                except StaleElementReferenceException as e:
                    print('lost element reference', tweet)

        except NoSuchElementException:
            print('no tweets on this day')

        start = increment_day(start, 1)

```

```
    ################ Show the condition when scraping is done ################
    try:
        with open(twitter_ids_filename) as f:
            all_ids = ids + json.load(f)
            data_to_write = list(set(all_ids))
            print('tweets found on this scrape: ', len(ids))
            print('total tweet count: ', len(data_to_write))
    except FileNotFoundError:
        with open(twitter_ids_filename, 'w') as f:
            all_ids = ids
            data_to_write = list(set(all_ids))
            print('tweets found on this scrape: ', len(ids))
            print('total tweet count: ', len(data_to_write))

    with open(twitter_ids_filename, 'w') as outfile:
        json.dump(data_to_write, outfile)

    print('all done here')
    driver.close()
    ############################################## Getting Tweets IDs ##############################################

    ###################################### Create and Zip Files ######################################
    with open(twitter_ids_filename) as f:
        ids = json.load(f)

    print('total ids: {}'.format(len(ids)))

    all_data = []
    start = 0
    end = 100
    limit = len(ids)
    i = math.ceil(limit / 100)

    for go in range(i):
        print('currently getting {} - {}'.format(start, end))
        sleep(6)  # needed to prevent hitting API rate limit
        id_batch = ids[start:end]
        start += 100
        end += 100
        tweets = api.statuses_lookup(id_batch)
        for tweet in tweets:
            all_data.append(dict(tweet._json))

    print('metadata collection complete')
    print('creating master json file')
    with open(output_file, 'w') as outfile:
        json.dump(all_data, outfile)

    print('creating ziped master json file')
    zf = zipfile.ZipFile('{}.zip'.format(user.split("\"")[1]+start_date), mode='w')
    zf.write(output_file, compress_type=compression)
    zf.close()
    ###################################### Create and Zip Files ######################################
```

```
119
120     ################################### Get Results ###################################
121     results=[]
122     with open(output_file) as json_data:
123         data = json.load(json_data)
124         for entry in data:
125             t = {
126                 "text": entry["text"],
127                 "Likes": entry["favorite_count"],
128                 "Retweet_count": entry["retweet_count"],
129                 "is_retweet": is_retweet(entry)
130             }
131             results.append(t)
132         display(results)
133
134     print('creating minimized json master file')
135     with open(output_file_short, 'w') as outfile:
136         json.dump(results, outfile)
137     ################################### Get Results ###################################
138
139     ################################### Get Results DataFrame ###################################
140     scraping_twitter_data = pd.DataFrame(data=results)
141     scraping_twitter_data['SA'] = np.array([analize_sentiment(tweet) for tweet in scraping_twitter_data['text']])
142     scraping_twitter_data['SA_Likes']=scraping_twitter_data['SA']*scraping_twitter_data['Likes']
143     scraping_twitter_data['SA_RTs']=scraping_twitter_data['SA']*scraping_twitter_data['Retweet_count']
144     scraping_twitter_data['SA_is_retweet']=scraping_twitter_data['SA']*scraping_twitter_data['is_retweet']
145     #save the result
146     scraping_twitter_data.to_csv('{}.csv'.format(keywards.split("\"")[1]+'_'+start_date),index=False)
147
148     TweetsNum=len(scraping_twitter_data)
149     Total_Likes=sum(scraping_twitter_data['Likes'])
150     Total_RTs=sum(scraping_twitter_data['Retweet_count'])
151     Sum_SA_Likes=sum(scraping_twitter_data['SA_Likes'])
152     Sum_SA_RTs=sum(scraping_twitter_data['SA_RTs'])
153     Sum_SA_is_retweet=sum(scraping_twitter_data['SA_is_retweet'])
154     Num_is_RTs=len(scraping_twitter_data[scraping_twitter_data['is_retweet']==True])
155     Positive=len(scraping_twitter_data[scraping_twitter_data['SA']==1])/TweetsNum
156     Negative=len(scraping_twitter_data[scraping_twitter_data['SA']==-1])/TweetsNum
157     ################################### Get Results DataFrame ###################################
158
159     return (TweetsNum,Total_Likes,Total_RTs,Sum_SA_Likes,Sum_SA_RTs,Sum_SA_is_retweet,Num_is_RTs,Positive,Negative)
```

Exhibit N





DATA CLEANING

Based on two raw dataset we collected, we aimed to create a dataset with all the variables we need, which includes *Movie Name, Genre, Director, Actors, Since Date, str(Since Date), Until Date, str(Until Date), Year, Month, Day, Overall Rating, Votes, Weekly Gross, Percentage Change, Cumulative Revenue, Average Revenue per Movie Theater,* and *Week Numbers.*

First, we join two datasets with Movie Name. However, we discovered the movie names of them are slightly different. Thus, we target the null data after joining them and change the movie names manually.

Second, we re-decode the Date and split it with "?", writing a function to transfer {Jan, Feb, Mar, ….} to {01, 02, 03, …..}, combining with Releasing Year to have a column of str(Since Date). Then we drop the Date column, tuning the week_number all starting from week 1 and insert 8 more rows for each movie (from week -7 to 0).

Next, we transfer the str(Since Date) of each movie in first week to Datetime, and plus (week time-1)*deltatime(7) as Since Date of the rest of other rows, adding deltatime(7) to have the Until Date and the str(Until Date). (*In order to access the twitter data*).

After cleaning, transferring data structure and filling the nulls, we finally have our data as following:

revenue_cleanned.iloc[:,0:10]

	index	Movie_Name	Weekly_Gross	Percentage_Change	Avg	Gross_to_Date	Week_Number	Year	since_str	since_date
0	0	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-7.0	2016	2016-01-15	2016-01-15 00:00:00
1	1	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-6.0	2016	2016-01-22	2016-01-22 00:00:00
2	2	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-5.0	2016	2016-01-29	2016-01-29 00:00:00
3	3	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-4.0	2016	2016-02-05	2016-02-05 00:00:00
4	4	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-3.0	2016	2016-02-12	2016-02-12 00:00:00
5	5	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-2.0	2016	2016-02-19	2016-02-19 00:00:00
6	6	10 Cloverfield Lane	0.0	0.00	0.0	0.0	-1.0	2016	2016-02-26	2016-02-26 00:00:00
7	7	10 Cloverfield Lane	0.0	0.00	0.0	0.0	0.0	2016	2016-03-04	2016-03-04 00:00:00
8	8	10 Cloverfield Lane	32676639.0	0.00	9636.0	32676639.0	1.0	2016	2016-03-11	2016-03-11 00:00:00
9	9	10 Cloverfield Lane	17334158.0	-0.47	5058.0	50010797.0	2.0	2016	2016-03-18	2016-03-18 00:00:00
10	10	10 Cloverfield Lane	8802263.0	-0.49	3141.0	58813060.0	3.0	2016	2016-03-25	2016-03-25 00:00:00
11	11	10 Cloverfield Lane	6162480.0	-0.30	2454.0	64975540.0	4.0	2016	2016-04-01	2016-04-01 00:00:00
12	12	10 Cloverfield Lane	3759141.0	-0.39	1993.0	68734681.0	5.0	2016	2016-04-08	2016-04-08 00:00:00
13	13	10 Cloverfield Lane	1440070.0	-0.62	1327.0	70174751.0	6.0	2016	2016-04-15	2016-04-15 00:00:00
14	14	10 Cloverfield Lane	774997.0	-0.46	1303.0	70949748.0	7.0	2016	2016-04-22	2016-04-22 00:00:00

revenue_cleanned.iloc[:,10:]

	until_date	until_str	Month	Day	Genre	Director	Actors	Rating	Revenue	Runtime	Voting	Metascore
0	2016-01-22	2016-01-22	1	15	Drama,Horror,Mystery	Dan Trachtenberg	John Goodman, Mary Elizabeth Winstead, John Ga...	7.2	71.90	104.0	192968.0	76.0
1	2016-01-29	2016-01-29	1	22	Drama,Horror,Mystery	Dan Trachtenberg	John Goodman, Mary Elizabeth Winstead, John Ga...	7.2	71.90	104.0	192968.0	76.0
2	2016-02-05	2016-02-05	1	29	Drama,Horror,Mystery	Dan Trachtenberg	John Goodman, Mary Elizabeth Winstead, John Ga...	7.2	71.90	104.0	192968.0	76.0
3	2016-02-12	2016-02-12	2	5	Drama,Horror,Mystery	Dan Trachtenberg	John Goodman, Mary Elizabeth Winstead, John Ga...	7.2	71.90	104.0	192968.0	76.0
4	2016-02-19	2016-02-19	2	12	Drama,Horror,Mystery	Dan Trachtenberg	John Goodman, Mary Elizabeth Winstead, John Ga...	7.2	71.90	104.0	192968.0	76.0

revenue_cleanned.describe()

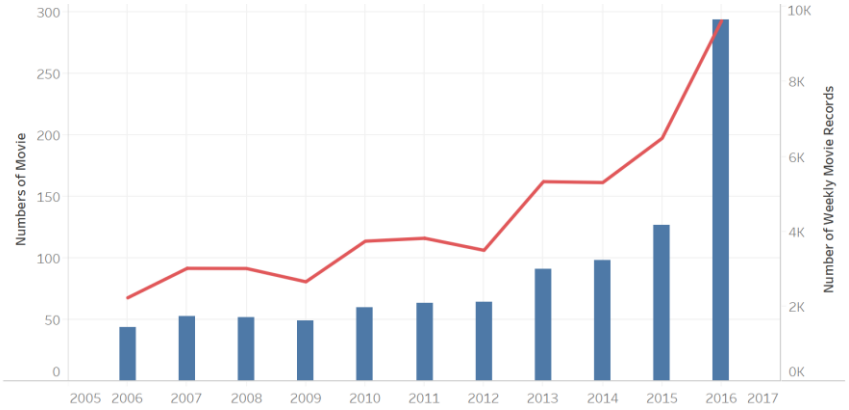
	index	Weekly_Gross	Percentage_Change	Avg	Gross_to_Date	Week_Number	Year	Month	Day	Rating	Revenue	Runtime	Voting	Metascore
count	19046.000000	1.904600e+04	19046.000000	19046.000000	1.904600e+04	19046.000000	19046.000000	19046.000000	19046.000000	19046.000000	19046.000000	19046.000000	1.904600e+04	19046.000000
mean	2021.817599	3.773839e+06	0.194034	2744.214901	6.133204e+07	4.632994	2012.419511	6.588733	15.725402	6.914323	99.361009	115.769820	2.198044e+05	62.848525
std	1287.203638	1.406337e+07	39.518762	11411.761148	1.021168e+08	8.137615	3.172209	3.424002	8.810999	0.854459	114.066506	18.657658	2.106652e+05	58.928399
min	0.000000	0.000000e+00	-1043.000000	0.000000	0.000000e+00	-7.000000	2005.000000	1.000000	1.000000	1.900000	0.000000	80.000000	9.600000e+01	0.000000
25%	954.000000	0.000000e+00	-0.420000	0.000000	0.000000e+00	-2.000000	2010.000000	4.000000	8.000000	6.400000	24.990000	102.000000	7.526200e+04	50.000000
50%	1906.000000	8.584900e+04	-0.050000	956.000000	1.394803e+07	4.000000	2013.000000	7.000000	16.000000	7.000000	59.890000	113.000000	1.576060e+05	62.000000
75%	2906.750000	1.058206e+06	0.000000	2157.000000	8.148854e+07	10.000000	2015.000000	10.000000	23.000000	7.500000	134.520000	126.000000	3.089810e+05	74.000000
max	5200.000000	3.908561e+08	3810.850000	616215.000000	9.366622e+08	51.000000	2017.000000	12.000000	31.000000	9.000000	936.630000	191.000000	1.791916e+06	2009.000000

revenue_cleanned.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19046 entries, 0 to 19045
Data columns (total 22 columns):
index              19046 non-null int64
Movie_Name         19046 non-null object
Weekly_Gross       19046 non-null float64
Percentage_Change  19046 non-null float64
Avg                19046 non-null float64
Gross_to_Date      19046 non-null float64
Week_Number        19046 non-null float64
Year               19046 non-null int64
since_str          19046 non-null object
since_date         19046 non-null object
until_date         19046 non-null object
until_str          19046 non-null object
Month              19046 non-null int64
Day                19046 non-null int64
Genre              19046 non-null object
Director           19046 non-null object
Actors             19046 non-null object
Rating             19046 non-null float64
Revenue            19046 non-null float64
Runtime            19046 non-null float64
Voting             19046 non-null float64
Metascore          19046 non-null float64
dtypes: float64(10), int64(4), object(8)
memory usage: 3.2+ MB
```

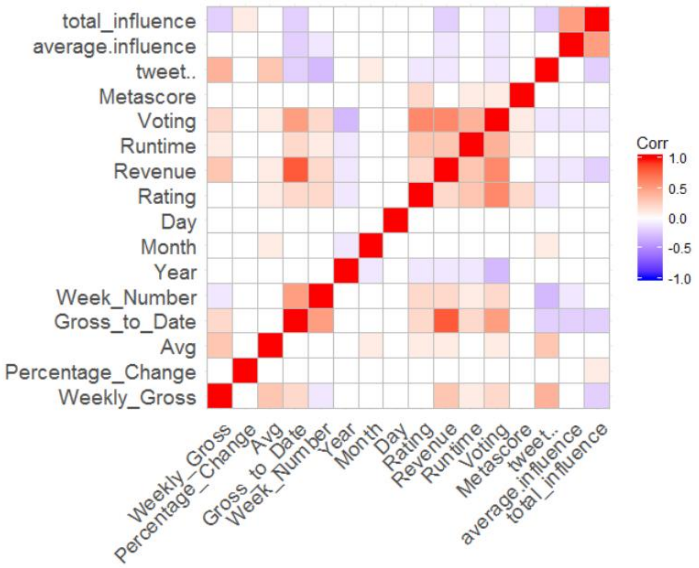
DATA EXPLORATION

1. Total number of released movies increased from year 2006 to 2016



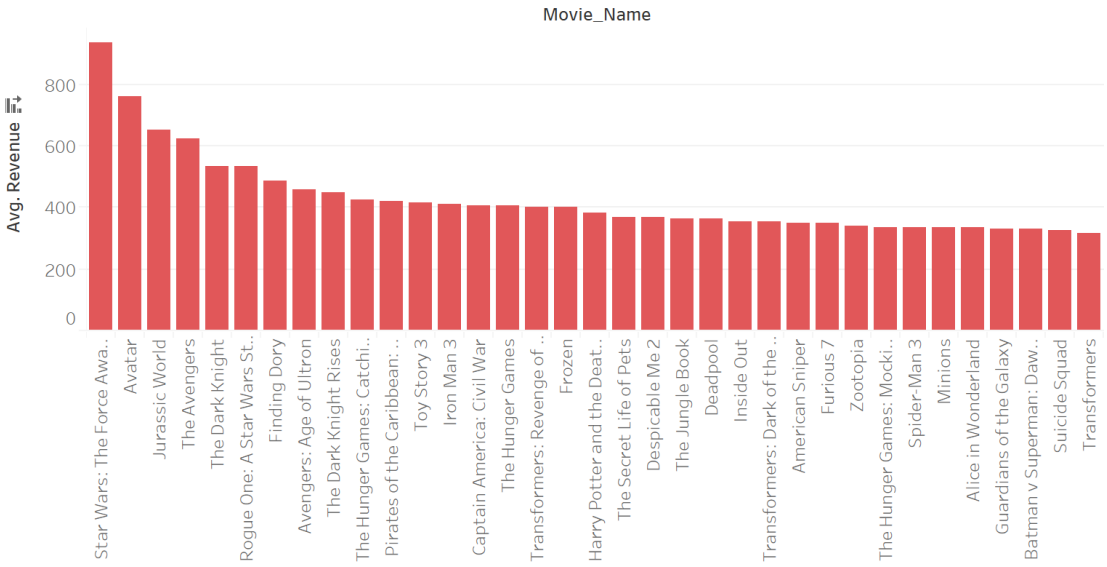
2. Below diagrams indicate the strength of the correlation between each variable. (Strongest: +1 to the lowest: -1)

The diagram indicates the correlation between each variable including the total week before and after the movie released. In the correlation matrix, we discovered that rating, run time and numbers of tweets are positively related to Movie Weekly Gross, and week number is negatively related to weekly gross.

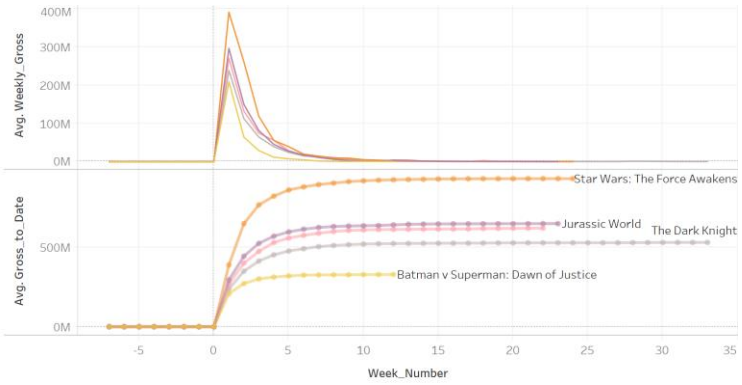


3. Revenue ranking of the 1,000 most popular movies on IMDB from year 2006 till year 2016

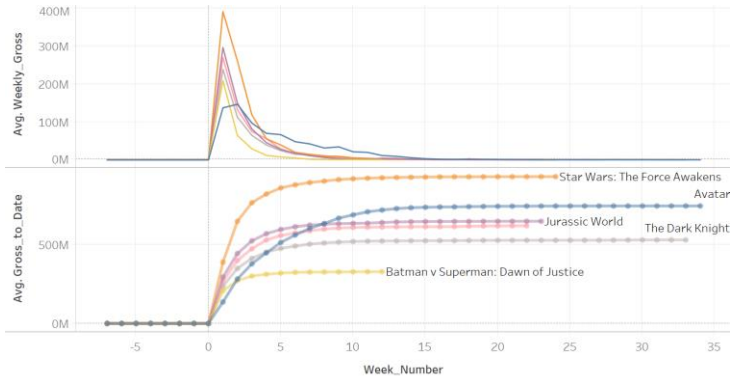
Sort the movie based on their total revenue, we can figure out that the top five movies were Star War, Avatar, Jurassic world, The Avengers and the Dark Knight.



4.
After plotting their weekly gross with their wee number, we can observe that most of the weekly gross follows a trend, which shows that the revenue of the first week will be the highest and decrease exponentially based on weeks. It means that the total revenue of a movie might highly be affected by the first week revenue.

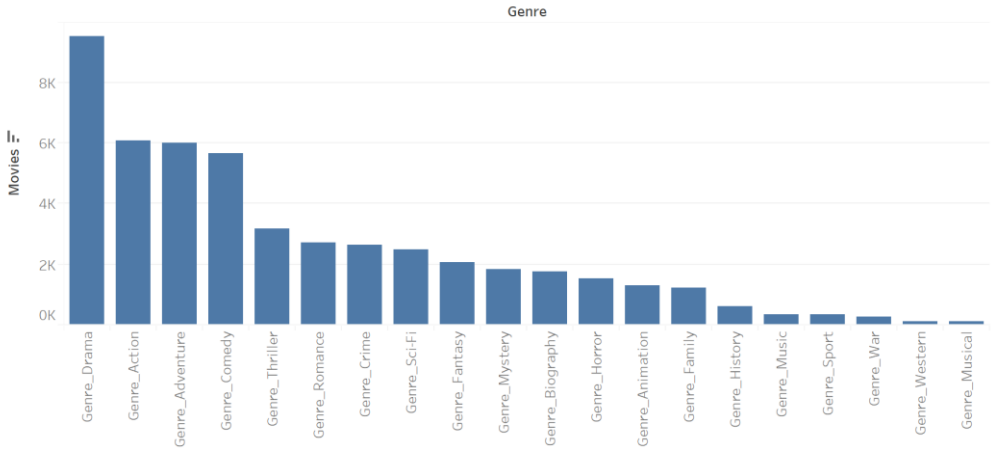


However, we also observed that Avatar, from James Francis Cameron, had the highest revenue at the second week and strongly earning for more than 33 weeks. We are trying to understand what actually affected it. Maybe we can increase the total revenue by increasing the releasing duration.



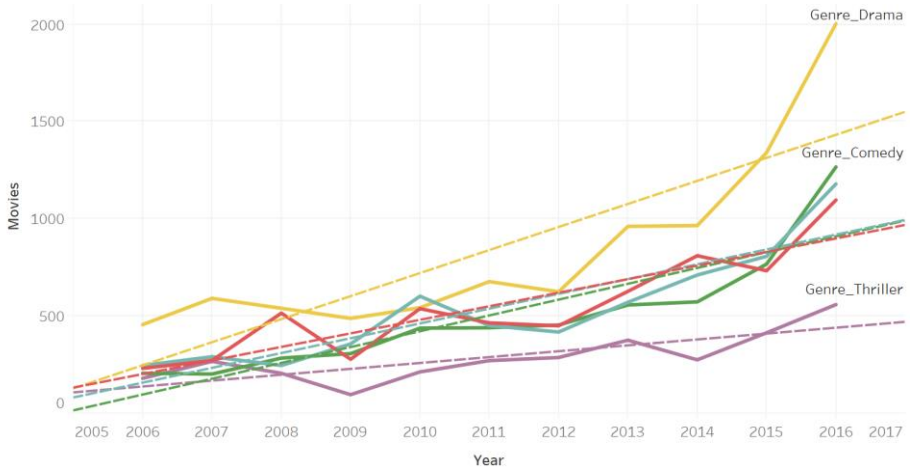
5. Ranking of the total number of genres from year 2006 till year 2016

The top five genres are Drama, Action, Adventure, Comedy, and Thriller. It actually makes sense because every movie is labeled with three Genres. If they are related to story, one of their label will be marked as Drama.



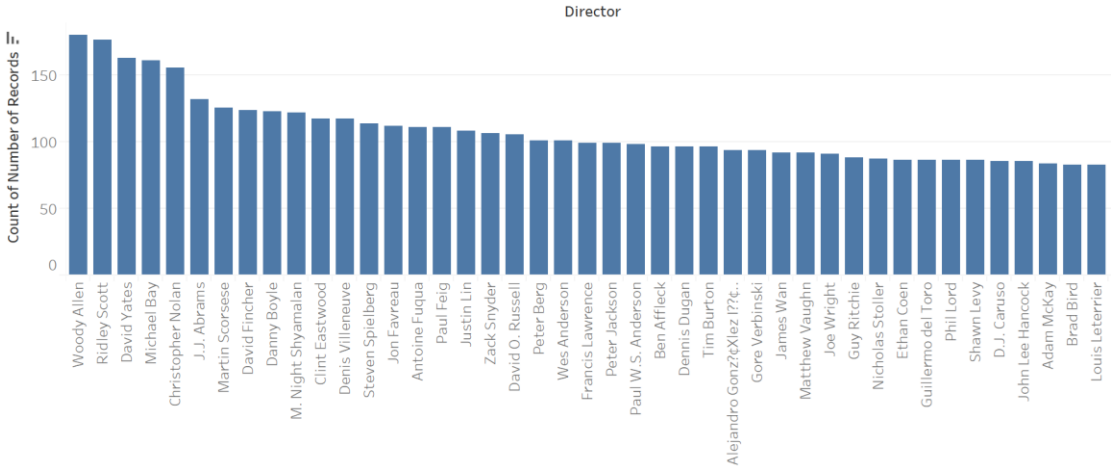
6. Annual total number of genres from year 2006 till year 2016

As we can see in the following graph, the frequency of Drama is always higher, while Action, Adventure, and Comedy remains slightly the same. In this graph, we can only conclude that Action, Adventure, and Comedy are relatively popular to the movie producers.

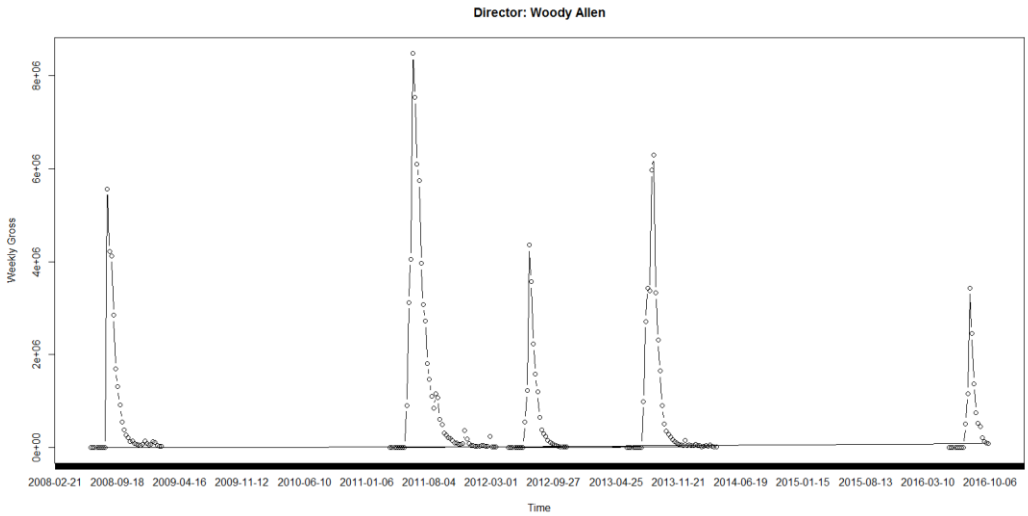


7. Director Ranking

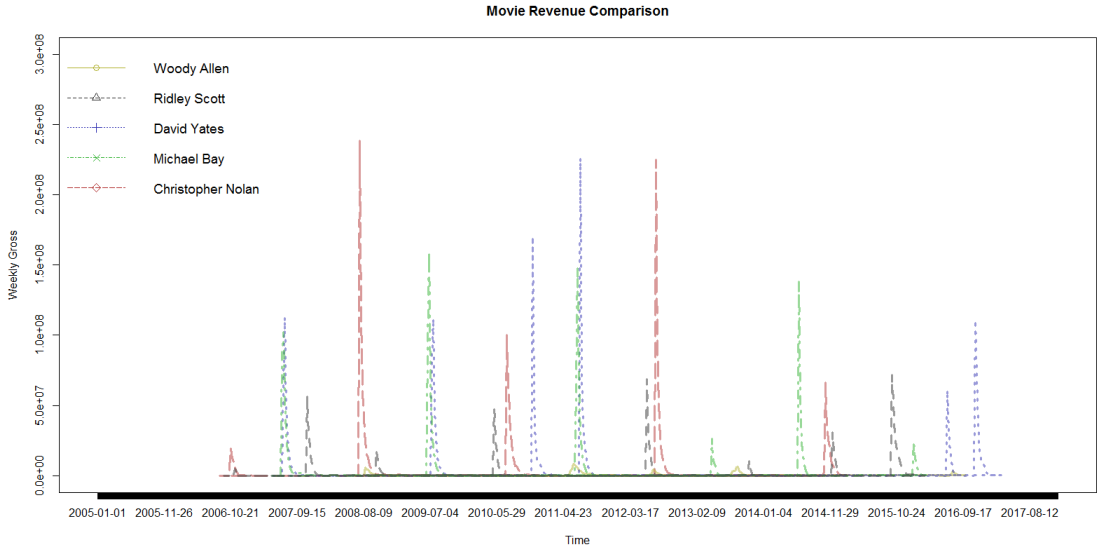
The top five director whose movies were most frequently on theater were Woody Allen (Midnight in Paris), Ridley Scott (Alien), David Yates (Harry Potter), Michael Bay (Transformer) and Cristopher Nolan (Inception).



We tried to plot the director whose movie has the highest records, and discovered that he had five movies from 2006 to 2016. The one who had highest revenue of them is midnight in Paris. It had over 151 million dollars gross and was on the theater for more than 44 weeks.

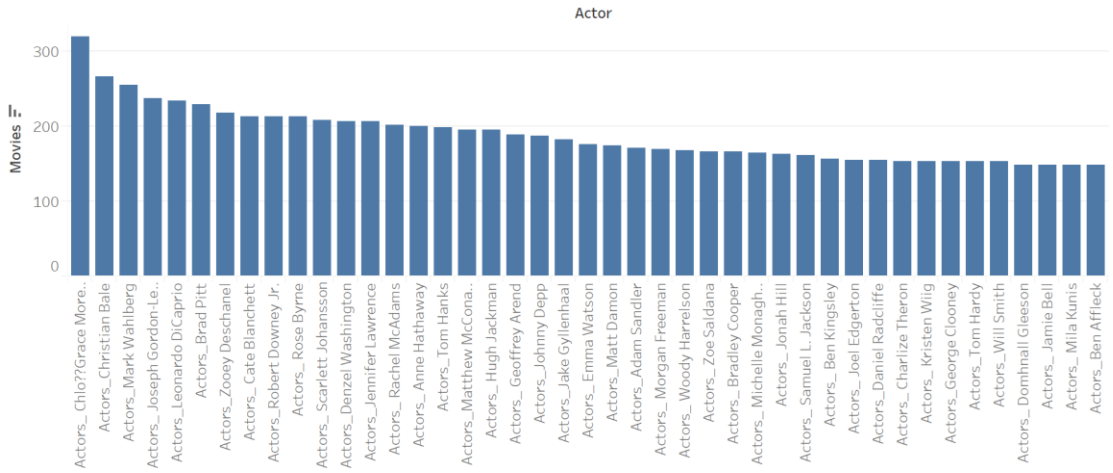


However, when plot the rest of top five directors (as the following graph), we found out Woody Allen's movie had the lowest revenue among these directors. In this case, we can state that the duration of movie on theater might not be the only factors that affect revenues. The second observation n this graph is that we discover that some of the directors were releasing their movie at the same time, and not only once. David Yates and Michael Bay releasing their movies at a similar time triple time, which was an interesting thing. On the other hand, Christopher Nolan chose a separate time. We also curious about how the releasing time will affect movie revenue.

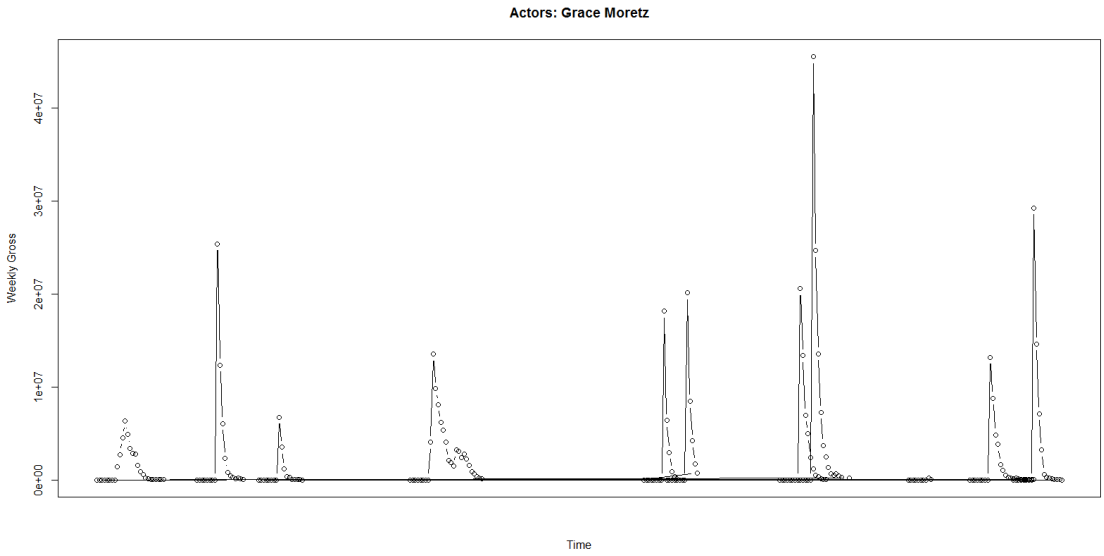


8. Actor ranking

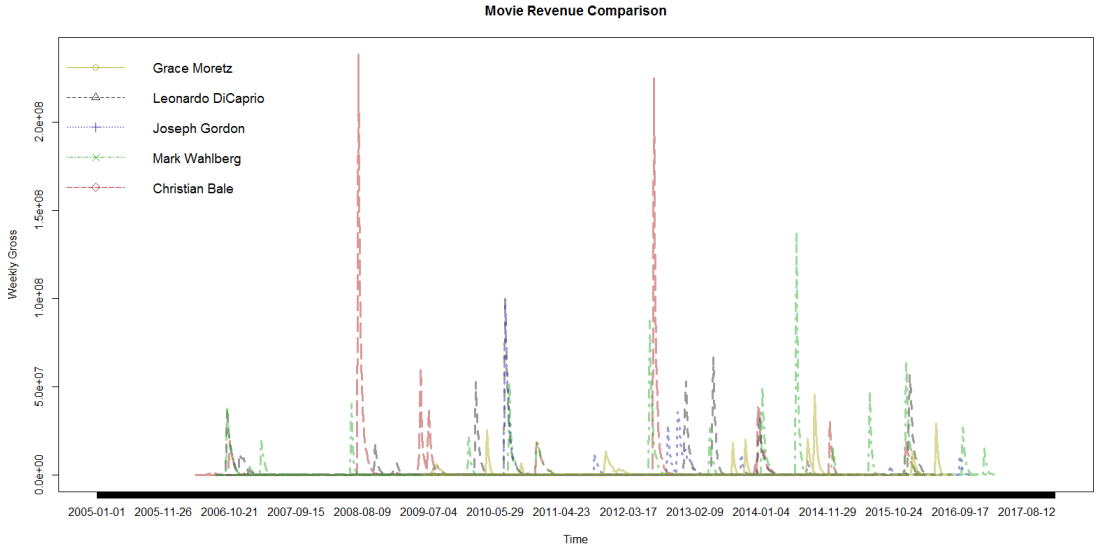
The top five actors who appeared most frequently were Chloë Grace Moretz, Christian Bale, Mark Wahlberg, Joseph Gordon, and Leonardo DiCaprio. Each of them was reasonable to be in the top and was in our expectation, except Chloë Grace Moretz.



Initially, we thought it was the wrong data; however, when we actually search her, we were surprised that she played a role in more than twenty-two movies and TV episodes during 2006 to 2016. (Although she only appeared 12 times in our dataset) With this information, we then realized that some actors are very important in this industry, even if they did not always play an important role.

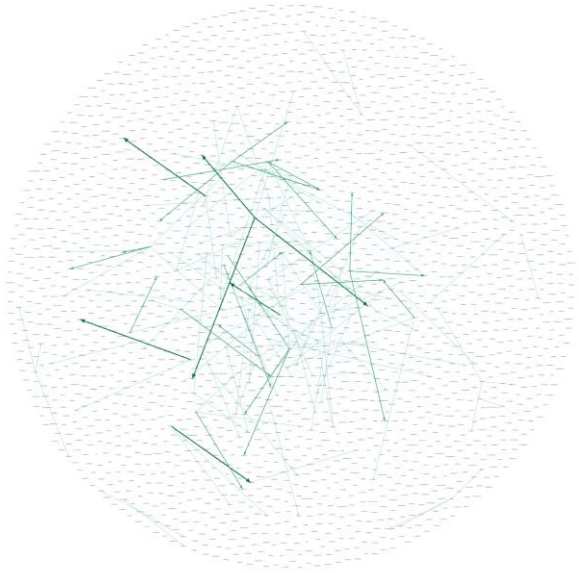


Of course, the movie she participated might not always as high as those movies with some of the most famous actors like Christian Bale or Leonardo DiCaprio.



9. Social Network between Director and Actor:

To evaluate the strength of the relationship between director and actor, we based on the social network diagram that generated and exported from the Gelphi application to see whether the director collaborated more frequently with certain actor in the movie. The data that we used only focus on the 1,000 most popular movies on IMDB from year 2006 till year 2016. Following is the whole view of the social network diagram. The thicker the line between the director and the actor is, the more strong and intense relationship between them.



DATA FEATURING

The quality of the dataset and the selective of those features determine the performance of the Machine Learning Model. When we are training the model, we need to do some preprocessing to improve the accuracy of total prediction. The purpose of those preprocessing is to enforce the model to achieve convergence and thus increase the accuracy.

One Hot Encoding

we need to use the one hot encoding to improve the overall performance in order to proceed the machine learning model in better accuracy. However, the Director and Actors have a large number of different categories (20 Genres, 660 Directors, and 1990 Actors) and it might create the curse of dimensionality. Therefore, we need to use get dummy function in the analysis process to change the categorical features into dummy variable and narrowing down the dimensions.

We failed to use the get dummy function because there are several variables in the columns of genre and actors. (Shown as follows)

```
#test Gerne
df_Genre=revenue_cleanned[['Movie_Name','Week_Number','Genre']][revenue_cleanned['Week_Number']==1]
df_Genre[df_Genre['Movie_Name']=='10 Cloverfield Lane']
```

	Movie_Name	Week_Number	Genre
8	10 Cloverfield Lane	1.0	Drama,Horror,Mystery

```
#test Actors
df_Actors=revenue_cleanned[['Movie_Name','Week_Number','Actors']][revenue_cleanned['Week_Number']==1]
df_Actors[df_Actors['Movie_Name']=='10 Cloverfield Lane']
```

	Movie_Name	Week_Number	Actors
8	10 Cloverfield Lane	1.0	John Goodman, Mary Elizabeth Winstead, John Ga...

Therefore, we firstly separated the columns with different variables into diverse rows and create two data frames as the following graph.

	Movie_Name	Week_Number	Genre
0	(500) Days of Summer	-20.0	Comedy
1	(500) Days of Summer	-19.0	Drama
2	(500) Days of Summer	-18.0	Romance
3	10 Cloverfield Lane	-20.0	Drama
4	10 Cloverfield Lane	-19.0	Horror
5	10 Cloverfield Lane	-18.0	Mystery
6	10 Years	-20.0	Comedy
7	10 Years	-19.0	Drama
8	10 Years	-18.0	Romance
9	12 Years a Slave	-20.0	Biography
10	12 Years a Slave	-19.0	Drama
11	12 Years a Slave	-18.0	History

	Movie_Name	Week_Number	Actors
0	(500) Days of Summer	-20.0	Zooey Deschanel
1	(500) Days of Summer	-19.0	Joseph Gordon-Levitt
2	(500) Days of Summer	-18.0	Geoffrey Arend
3	(500) Days of Summer	-17.0	Chlo??Grace Moretz
4	10 Cloverfield Lane	-20.0	John Goodman
5	10 Cloverfield Lane	-19.0	Mary Elizabeth Winstead
6	10 Cloverfield Lane	-18.0	John Gallagher Jr.
7	10 Cloverfield Lane	-17.0	Douglas M. Griffin
8	10 Years	-20.0	Channing Tatum
9	10 Years	-19.0	Rosario Dawson
10	10 Years	-18.0	Chris Pratt
11	10 Years	-17.0	Jenna Dewan Tatum

After doing so, we used get_dummy() to have new columns and then combine them together.

Movie_Name	Genre_Action	Genre_Adventure	Genre_Animation	Genre_Biography	Genre_Comedy	Genre_Crime	Genre_Drama	Genre_Family	Genre_Fantasy	...
(500) Days of Summer	0	0	0	0	1	0	1	0	0	...
10 Cloverfield Lane	0	0	0	0	0	0	1	0	0	...

Normalization

After doing the one hot encoding, we used standard scaling to make our data more manageable in order to preserve the significance of the data set. By doing this approach can enables us to make all the variables into the same scale, and also preserve their variance without affecting the relative importance. The alternative approach to use Z-score normalization is called Min-Max scaling. In the method, we use the Z-score normalization method to scale the fixed range of data into 0 to 1. First, we import the pandas as pd and we import the MinMaxScaler from the sklearn preprocessing. As you can see the exhibit 4., we can conclude that the df_test.csv data has been transformed in to the dftest2 as a new scaler as the exhibits 5 shows below:

Exhibit 4.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
```

Exhibit 5.

```
dfTest2
```

```
array([[  0.00000000e+00,   2.14880971e-01,   0.00000000e+00, ...,
          3.78297661e-02,   3.86584234e-02,   4.75582752e-01],
       [  0.00000000e+00,   2.14880971e-01,   0.00000000e+00, ...,
          3.78297661e-02,   3.06411678e-02,   4.75473461e-01],
       [  0.00000000e+00,   2.14880971e-01,   0.00000000e+00, ...,
          3.78297661e-02,   3.09633407e-02,   4.75574519e-01],
       ...,
       [  1.95391626e-04,   2.14829465e-01,   1.20250237e-03, ...,
          2.78745645e-02,   3.86152851e-02,   4.75145063e-01],
       [  7.97915235e-05,   2.14759418e-01,   1.01263358e-03, ...,
          2.78745645e-02,   3.30018612e-02,   4.75042483e-01],
       [  4.00224068e-05,   2.14777960e-01,   9.39607118e-04, ...,
          2.78745645e-02,   4.73801459e-02,   4.75191603e-01]])
```

Exhibit 6.

```
dftrans
```

	Weekly_Gross	Percentage_Change	Avg	Gross_to_Date	Rating	Runtime	Voting	Metascore	tweet #	average influence
0	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.038658	0.475583
1	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.030641	0.475473
2	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.030963	0.475575
3	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.056501	0.475458
4	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.120996	0.475613
5	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.251491	0.475579
6	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.519395	0.475671
7	0.000000	0.214881	0.000000	0.000000	0.746479	0.216216	0.107640	0.037830	0.474472	0.475652
8	0.083603	0.214881	0.015637	0.034886	0.746479	0.216216	0.107640	0.037830	0.633300	0.475595
9	0.044349	0.214784	0.008208	0.053393	0.746479	0.216216	0.107640	0.037830	0.862448	0.472265
10	0.022520	0.214780	0.005097	0.062790	0.746479	0.216216	0.107640	0.037830	0.516855	0.472491
11	0.015767	0.214819	0.003982	0.069369	0.746479	0.216216	0.107640	0.037830	0.487285	0.474339
12	0.009618	0.214801	0.003234	0.073383	0.746479	0.216216	0.107640	0.037830	0.310078	0.473959

Principle Component Analysis

After the scaler process, we used the Principle Component Analysis (PCA) to turn our data into a lower dimensional data set. To elaborate, the PCA can extracting important data variables from a higher dimensional variables into a lower dimensional data set. By running the PCA method in fit_transform df17, it allows us to use those important features in the data set to generate better information to make a better decision on training our data sets (see new data set as below)

Exhibit 7.

```
>>> from sklearn.decomposition import PCA
```

```
pca=PCA(n_components=500)  #'mle'
newData=pca.fit_transform(df17)
newData

#>>> pca=PCA(n_components='mle')
#>>> newData=pca.fit_transform(data)
#>>> newData
```

```
array([[-0.64376523, -0.35389511,  0.18824326, ..., -0.06114093,
        -0.04607185, -0.0073864 ],
       [-0.64371432, -0.35381012,  0.18796501, ..., -0.06114766,
        -0.04605395, -0.00738433],
       [-0.64371643, -0.35381353,  0.18797622, ..., -0.06114735,
        -0.04605476, -0.00738444],
       ...,
       [-0.22051062,  1.05214061,  0.31491842, ...,  0.05210384,
        -0.06901487, -0.05559433],
       [-0.22047475,  1.05220021,  0.31472429, ...,  0.05209866,
        -0.06900245, -0.05559464],
       [-0.22056595,  1.05204781,  0.31522355, ...,  0.05211049,
        -0.06903452, -0.05559899]])
```

As we can see through exhibit 7, we can learn that the variables has been cut into 500 variables in the data set and we can conclude a 82% significance in the PCA outputs (See exhibits 8.). As a result, the visualization becomes much more meaningful. And it is the time to do our machine learning models after the data featuring process.

Exhibit 8.

```
sum(pca.explained_variance_ratio_[1:500])
```

```
0.81677224292409178
```

```
pca.explained_variance_ratio_
```

```
array([ 0.07610483,  0.0507168 ,  0.03423505,  0.02583257,  0.02354179,
        0.02009629,  0.0169193 ,  0.01677643,  0.01334851,  0.01222403,
        0.01129446,  0.00998195,  0.00858235,  0.00789693,  0.00602639,
        0.00548474,  0.00516855,  0.00387386,  0.00357904,  0.00352396,
        0.00345002,  0.00333386,  0.00321747,  0.00312714,  0.00300535,
        0.00298251,  0.00287133,  0.00284169,  0.00279777,  0.00276247,
        0.00274223,  0.0026985 ,  0.00267794,  0.00265395,  0.00262256,
        0.00259282,  0.00258184,  0.00255668,  0.00252404,  0.00249778,
        0.00249017,  0.00245984,  0.00241825,  0.00240329,  0.00236652,
        0.00235797,  0.00234355,  0.00233208,  0.00231567,  0.00230323,
        0.00229236,  0.00224689,  0.00222553,  0.00221375,  0.00219138,
        0.00217736,  0.00214816,  0.00214267,  0.00213465,  0.00211933,
        0.00210924,  0.00210079,  0.00207857,  0.00206281,  0.00204499,
        0.00201689,  0.00200604,  0.00199955,  0.0019879 ,  0.00197984,
        0.00197207,  0.0019584 ,  0.00195256,  0.0019322 ,  0.00192895,
        0.00192221,  0.00190092,  0.00189581,  0.00188568,  0.00186114,
        0.00185411,  0.00185338,  0.00183342,  0.0018194 ,  0.00181489,
        0.00180512,  0.00179543,  0.00178556,  0.00177513,  0.00175821,
```

DATA MODELING

Machine Learning Model:

In this section, we divided the movie data set into two parts which are 80% of the movies treated as the training set and the rest of the test set as 20%. Furthermore, we imported the library from Sklearn to import machine learning models such as SVR, SVM, Random Forest Regressor, Gradient Boosting Regressor, Ada Boost Regressor, Decision Tree Regressor and Bagging Regressor to train our final version of movie dataset and get the RMSPE scores in order to compare and determine which Machine Learning Algorithm that best describes the accuracy of the prediction.

First, we split the data frame into train data and test data, and return the RMSPE's scores of each graph after the trained model. We return the after trained data with n trails and do the cross-validation in the train model list. We import the file after implementing the normalization and PCA methods. Therefore, the graph shows the new dimension data frame with the reduction of the computational costs and the error of parameter estimation (see figure 9.)

Figure 9.

	3	4	5	6	7	8	9	Movie_Name	Week_Number	Year	Month	Day	Revenue	NGross_to_Date	NWeekly_Gross
23851	0.111737	-0.035826	-0.722587	0.107126	0.237933	-0.873118	10 Cloverfield Lane	-7	2016	1	15	71.9	0.0	0.0
23815	0.111802	-0.035700	-0.722722	0.107031	0.237954	-0.873306	10 Cloverfield Lane	-6	2016	1	22	71.9	0.0	0.0
23671	0.112069	-0.035194	-0.723271	0.106646	0.238042	-0.874071	10 Cloverfield Lane	-5	2016	1	29	71.9	0.0	0.0
23708	0.112001	-0.035326	-0.723129	0.106745	0.238021	-0.873876	10 Cloverfield Lane	-4	2016	2	5	71.9	0.0	0.0
23620	0.112162	-0.035016	-0.723462	0.106512	0.238073	-0.874336	10 Cloverfield Lane	-3	2016	2	12	71.9	0.0	0.0

After proceeding the scaler and PCA process, we import the regressors as below and we want to compare the machine learning model between SVR(), Linear SVR(), Random Forest Regressor(), Decision Tree Regressor (max_depth=3) (see figure 10.)

Figure 10.

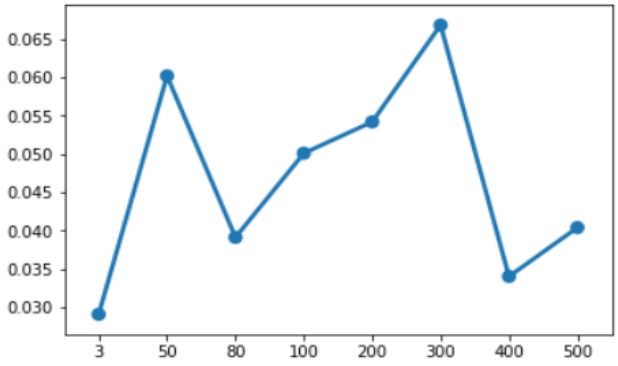
```
%matplotlib inline
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
#seaborn is a visulization for pandas
import seaborn as sns
from sklearn.model_selection import train_test_split
import datetime
# Importing Classifier Modules

#####################################################
#Classifier
#from sklearn.linear_model import LogisticRegression
#from sklearn.svm import SVC, LinearSVC
#from sklearn.neighbors import KNeighborsClassifier #KNN
#from sklearn.tree import DecisionTreeClassifier
#from sklearn.ensemble import RandomForestClassifier
#from sklearn.naive_bayes import GaussianNB
#from sklearn.linear_model import Perceptron
#from sklearn.linear_model import SGDClassifier
#from sklearn.ensemble import GradientBoostingClassifier
#####################################################

#####################################################
#Regressor
from sklearn.svm import SVR, LinearSVR
from sklearn.neighbors import KNeighborsRegressor #KNN
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
#from sklearn.naive_bayes import GaussianNB
#from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
```

Furthermore, we used the following test process to compare between Boosting Regressor and Bagging Regressor.

First, for AdaBoost regressor, we fitted the regressor on the original movie dataset. To elaborate, we needed to fit the additional copies of the regressor on the same data and the Ada boost regressor will adjust the weight of instances base on the error of the current prediction and improve the final estimation. Secondly, we used the voting algorithm to specifies the voting result of all the input data in the movie dataset and saw the pattern in the voting routine which would enhance our performance in the Machine Learning progress.

We set the AdaBoostingRegressor plus the RandomForest with several n_estimators in the RMSPE model and plot it as figure 11. As we can see, the Y-axis represents the RMSPE rate and the X-axis shows the n_estimators_list=[3,50,80,100,200,300,400,500]. Furthermore, we get the output that the estimators_list proves that 400 times as higher significant training estimators and we decide to use it as a standard point.

Figure 11.



As you can see in figure 12, we use Comparison Between Different Regressors (Ada-RandomForest) to run the estimated models to 400 times. We trained the data by comparison between different Regressors focusing on Ada-Random Forest. And the output shows a good

performance on Gradient-boosting and Ada-boosting because it performs the lowest RMSPE comparing to other machine learning models. Furthermore, we trained the data set adding Bagging Regressor with Random Forest in diverse 400_estimators in the model. Hence, we can learn from the figure 13. that the plot graph shows a lower RMSPE when the estimators are set to 50. Thus, we train the data set as 50 estimators and we still get the better performance in Ada-Boosting model according to the figure 14.

Figure 12.

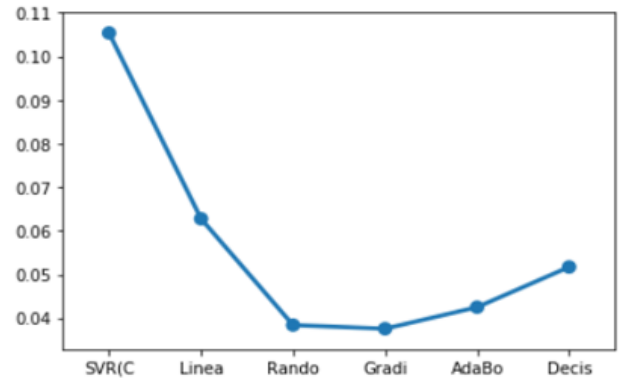


Figure 13.

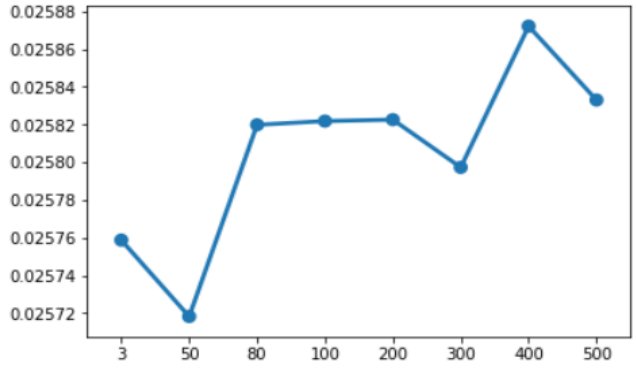
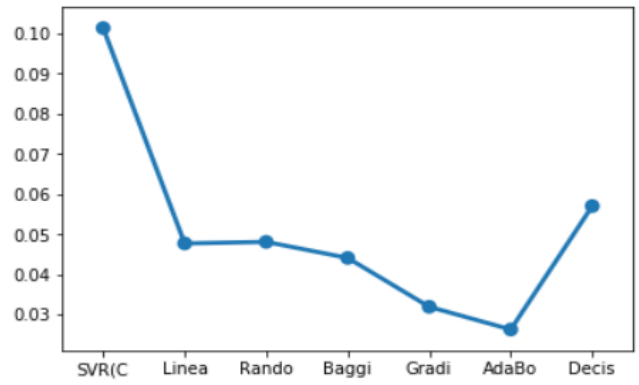


Figure 14.



As a result, we selected a movie from the movie dataset and built an RM model to plot the variance between the prediction and the real value. We trained the plot using function rm=AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),n_estimators=400). According to the output (figure 15.), we conclude that the model presents neither high bias and high variance when running the Ada boost Regressor(Random Forest). We recommend that in the future, movie producer can consider this performance with the Ada boost training in their decision making. That is to say; the Ada-boost regressor did minimize loss function by definite the minimize of the high bias. Also, when training the model, we should consider the trade-off between Bias/Variance Tradeoff to determine the ultimate model whether Overfitting or Underfitting in the Machine Learning process.

Figure 15.

```
plt.plot(range(0,len(RM_model)),RM_model,'r-',range(0,len(RM_model)),y_ttest,'go')
print(RMSPE(RM_model,y_ttest))
```

```
0.0236198662457
```

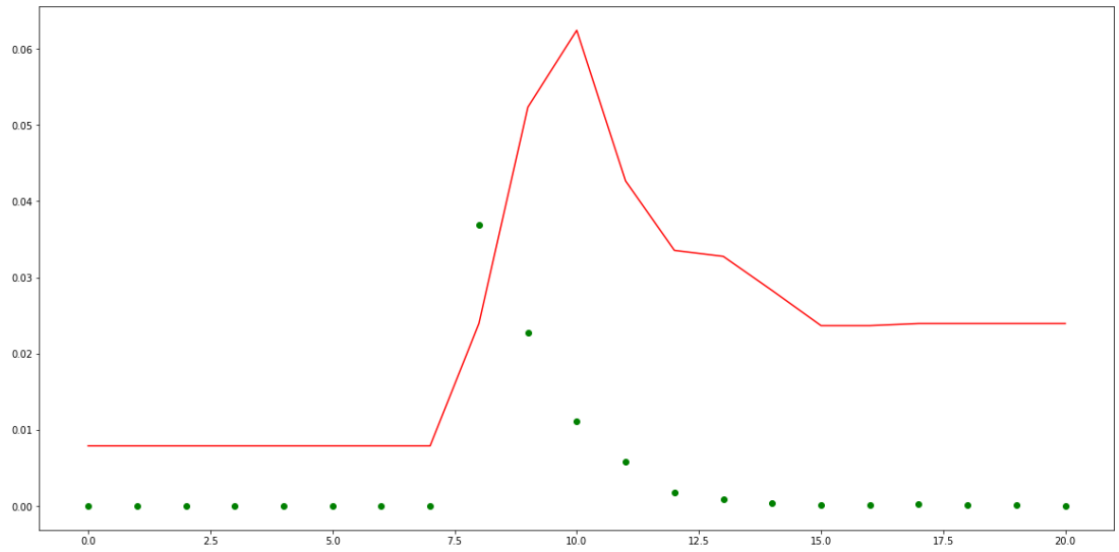
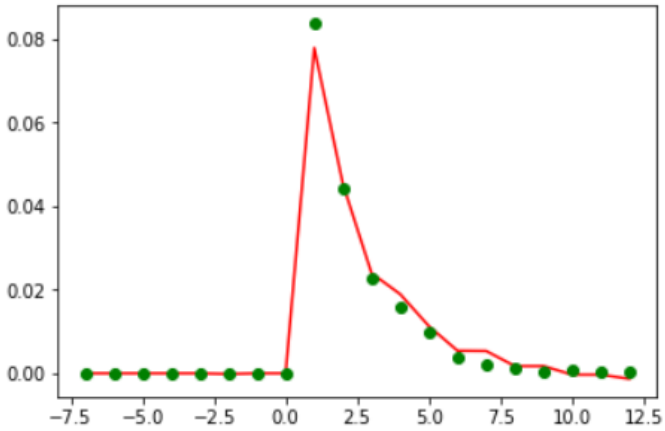
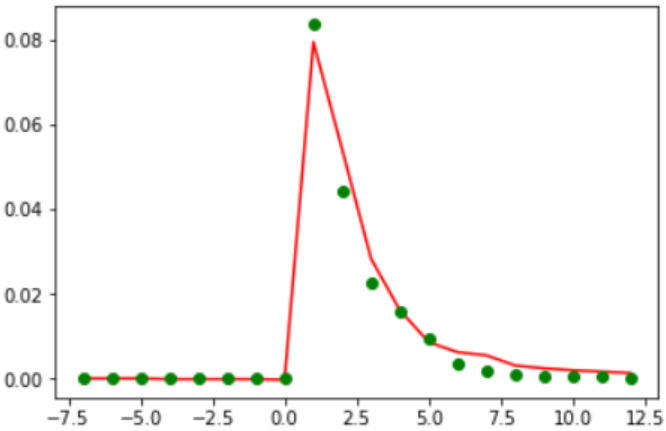


Exhibit N . Predict and Test Movie Graph

```
[<matplotlib.lines.Line2D at 0x1aa3f254438>,
 <matplotlib.lines.Line2D at 0x1aa3f2549b0>]
```





ANALYSIS

Business Value

1. **Possibility of accurately predict box office**

Based on the result of our prediction model, we can see the possibility of accurately predicting the box office for the upcoming and the released movies by using sentiment analysis on the tweets. Film company can based on the estimation comparing to further adjust their marketing plan such as marketing channel budget, the movie event.

2. **Influence the film demand**

To increase the revenue of the estimated box office, film company can change the demand for their film directly influencing the rate at which people tweet about. Such as how to increase the positive tweet discussion on twitter, how to increase the postive word of moth through twitter.

3. **Create hype before movie release**

Using twitter as one of the engaging channel for films to creating hype before movie release, a good premiere could get more attraction and exposure for the movie, most importantly, create synergy once the mine release on foreign countries.

Movie industry is a risky business. Just like the president of the Motion Picture Association of America (MPAA) once said, Not until the film opens in darkened theatre. No one can actually tell you how a movie is going to do in the marketplace. Which means that although it is important for film company to be able to accurately predict box office before they are released, the prediction is still an art just like movie itself due to the complexity of many factors behind what we see. However, the most important things that we learn from this practice is to understand how social media analysis can actually to be utilized to forecast future outcomes.

APPENDIX

REFERENCE

(n.d.). Scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. *Comparison of kernel ridge regression and SVR — scikit-learn 0.19.1 documentation.* earn.org/stable/auto_examples/plot_kernel_ridge_regression.html#sphx-glr-auto-examples-plot-kernel-ridge-regression-pyRetrieved from http://scikit-l

(n.d.). Scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. *Sklearn.preprocessing.StandardScaler — scikit-learn 0.19.1 documentation.* Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler.inverse_transform

(n.d.). Scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. *1.11. Ensemble methods — scikit-learn 0.19.1 documentation.* Retrieved from http://scikit-learn.org/stable/modules/ensemble.html#adaboost

(n.d.). Scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. *Sklearn.ensemble.AdaBoostRegressor — scikit-learn 0.19.1 documentation.* Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

(n.d.). The world’s leading software development platform · GitHub. *GitHub - bpb27/twitter_scraping: Grab all a user's tweets (and get past 3200 limit).* Retrieved from http://github.com/bpb27/twitter_scraping

(n.d.). (@yoyoyohamapi) on GitBook. *PCA · 斯坦福机器学习笔记.* Retrieved from http://yoyoyohamapi.gitbooks.io/mit-ml/content/%E7%89%B9%E5%BE%81%E9%99%8D%E7%BB%B4/articles/PCA.html