Jarvis_Project (9)

December 5, 2017

1 Data Visualization

1.1 Data Scraped, Downloaded & Cleaned

```
In [1]: import re, json, requests, seaborn, warnings
        warnings.filterwarnings( 'ignore' )
        import pandas as pd
        from bs4 import BeautifulSoup
        import numpy as np
        from matplotlib import pyplot as plt, rcParams
        %matplotlib inline
        from selenium import webdriver
        from selenium.webdriver.common.keys import Keys
        from selenium.common.exceptions import NoSuchElementException
```

1.1.1 Source of Movies

In the cell below: - I gather the urls for lists of the top 100 movies for each genre from rotten tomatoes. Note: Because I am only scraping movies from a top 100 list from rotten tomatoes, I am definitely gathering a biased selection of movies. Rotten tomatoes doesn't identify how they are ranking these movies as the "Top 100" so that could possibly introduce more bias into the data set. To help fix any bias that may be introduced by collecting my movies this way, I will be collecting a large number of movies, ~1000

```
In [2]: base_url = "https://www.rottentomatoes.com"
        action = "https://www.rottentomatoes.com/top/bestofrt/top_100_action__adventure_movies/"
        url_finder = re.compile( r"/top/bestofrt/top_100_.*$" )
        page_source = requests.get( action ).text
        soup = BeautifulSoup( page_source, "html.parser" )
        urls = soup.find_all( "a", href = url_finder )
        top = []
        for url in urls:
            top.append( base_url + url[ "href" ] )
```

1.1.2 Scraping Movie Data

In the cell below: - I go to each url that was gathered in the cell above - I get the name, content rating, and rotten tomatoes rating of each movie on the list.

```
In [39]: all_movies = []
         data = []
         for url in top:
             page_source = requests.get( url ).text
             soup = BeautifulSoup( page_source, "html.parser" )
             top_100 = soup.find( 'script', type = 'application/ld+json' )
             movie_data = json.loads( top_100.text )
             list_dict = movie_data[ "itemListElement" ]
             for top_movies in list_dict:
                 page_source = requests.get( top_movies[ "url" ] ).text
                 soup = BeautifulSoup( page_source, "html.parser" )
                 movie_data = json.loads( soup.find( 'script', type = 'application/ld+json' ).te
                 name = movie_data[ "name" ]
                 if name == "The Good, the Bad, the Weird (Joheun-nom, Nabbeun-nom, Isanghan-nom
                     name = "The Good, the Bad, the Weird"
                 if name not in all_movies:
                     try:    ratings = movie_data[ 'contentRating' ]
                     except: ratings = ""
                     try:    scorert = movie_data[ 'aggregateRating' ][ 'ratingValue' ]
                     except: scorert = np.nan
                     data.append( [ name, ratings, scorert ] )
                 all_movies.append( name )
```

1.1.3 Gathering Monetary Information on Each Movie

In the cell below: - I create a set of the movies name. - For each movie I search IMDB for the specific movie, and pick the first result. - I collect data for "Release Date", "Budget", "Opening Weekend", "Gross" - Any missing data is reported as NaN

IMDB is a reputable source for information, according to their website

> "we [IMDB] actively gather information from and verify items with studios and filmmakers".

I collect data on the release date, the budget for the movie, the amount of money spent on the movie during the opening weekend, and the gross profit made. Unfortunatley all these monetary amounts come in different currencies, so I will have to make this uniform and convert every currency to USD based on the date that the movie was released.

```
In [40]: base_url = "http://www.imdb.com/"
         browser = webdriver.Chrome()
         browser.get( base_url )
         for i in range( len( data ) ):
             # Get the search bar, type in some text, and press Enter.
             search_bar = browser.find_element_by_id( "navbar-query" )
             search_bar.clear() # Clear any pre-set text.
             search_bar.send_keys( data[ i ][ 0 ] ) # Searching for the name of the movie
             search_bar.send_keys( Keys.RETURN ) # Press Enter.
             soup = BeautifulSoup( browser.page_source, "html.parser" )
             table = soup.find( name = "table" )
```

```
            link = table.find_all( name = "td" )[ 1 ] # Choose first search result
            url = base_url + link.find( name = "a" )[ "href" ]
            browser.get( url ) # Get page for first movie result
            soup = BeautifulSoup( browser.page_source, "html.parser" )
            tab = soup.find_all( name = "h4" )
            B, O, G, R = False, False, False, False
            for j in tab:
                if j.text == "Release Date:" :
                    data[ i ].append( ( " " ).join( j.next_sibling.split() ) )
                    R = True
                if j.text == "Budget:":
                    data[ i ].append( ( " " ).join( j.next_sibling.split() ) )
                    B = True
                if j.text == "Opening Weekend:":
                    data[ i ].append( ( " " ).join( j.next_sibling.split( ) ) )
                    O = True
                if j.text == "Gross:":
                    data[ i ].append( ( " " ).join( j.next_sibling.split( ) ) )
                    G = True
            # Inserting NaNs for missing values
            if not R: data[ i ].insert( 3, np.nan )
            if not B: data[ i ].insert( 4, np.nan )
            if not O: data[ i ].insert( 5, np.nan )
            if not G: data[ i ].insert( 6, np.nan )
        browser.close()
```

1.1.4 Uncleaned Dataset

```
In [71]: n = len( data )
         columns = [ "Name", "Rating", "Score", "Release", "Budget", "Open", "Gross" ]
         df = pd.DataFrame( data, index = np.arange( n ), columns = columns )
         df.sample(5)
```

Out[71]:

	Name	Rating	Score	Release	Budget	Open	Gross
298	The Last Picture Show	R	100	22 October 1971 (USA)	$1,300,000	NaN	$29,133,000
727	Kung Fu Hustle	R	90	22 April 2005 (USA)	$20,000,000	$3,188,968 (Hong Kong)	$17,108,591
594	First Position	NR	93	10 August 2012 (Taiwan)	NaN	$48,024 (USA)	$1,100,000
844	The Normal Heart	R	94	25 May 2014 (USA)	NaN	NaN	NaN
568	The Autopsy of Jane Doe	R	87	21 December 2016 (USA)	NaN	251,128 (Italy)	NaN

1.1.5 Cleaning Data

In the cell below:

- I remove the phrases in parenthesis after the release data, telling where the movie was first released.
- I remove any commas at the end monetary amounts to make it easier to convert them to ints later.
- For each currency, I remove the symbol for the currency in place it in a new column that corresponds to opening weekend, and budget.
- Each date is formated uniformly with pandas.to_datetime()

Note: Because all gross profits are listed in USD I will not have to worry about that column

```
In [72]: df1 = df
         def remove_parenthesis( x ):
             x = [ str( i ).split( "(" )[ 0 ] for i in x ]
             return pd.Series( x )
         def remove_comma( x ):
             l = []
             for i in x:
                 l.append( ( "" ).join( str( i ).split( "," ) ) )
             return pd.Series( l )
         def _type( x ):
             if x == "nan":
                 return np.nan
             xs = x.split()
             if len( xs ) > 1:
                 return xs[ 0 ]
             else:
                 return x[ 0 ]
         def _int( x ):
             if x == "nan":
                 return np.nan
             xs = x.split()
             if len( xs ) > 1:
                 return int( xs[ 1 ] )
             else:
                 return int( x[ 1: ] )
         def _float( x ):
             return( float( x ) )

         df1      = df1.apply( remove_parenthesis, axis = 1 )
         df1      = df1.apply( remove_comma,       axis = 1 )
         df1[ 3 ] = df1[ 3 ].apply( lambda x: pd.to_datetime( str( x ) ) )
         df1[ 7 ] = df1[ 4 ].apply( _type )
         df1[ 8 ] = df1[ 5 ].apply( _type )
```

```
df1[ 2 ] = df1[ 2 ].apply( _float  )
df1[ 4 ] = df1[ 4 ].apply( _int  ) #column 4 units described by column 7
df1[ 5 ] = df1[ 5 ].apply( _int  ) #column 5 units described by column 8
df1[ 6 ] = df1[ 6 ].apply( _int  ) #column 6 is all in USD
```

1.1.6 Finding Data to Convert Currency

In the cell below: - I read in a dataset that I found that has values for the currency rates from dates from 1971 to 2017, for the exchange rate from USD to 26 different currencies

```
In [73]: e_r_columns = [ "Dates",   "",   "č", "BRL",  "CNY", "DKK", "INR", "JPY", "KRW", "MYR",
                         "CHF", "TWD", "THB", "VEB", "NBDI", "NMC", "NOI", "AUD", "NZD", "CAD"
         exchange_rates = pd.read_csv( "exchange_rates.csv", skiprows = [ 0, 1, 2, 4, 5, 6 ] )
         exchange_rates.columns = e_r_columns
         exchange_rates[ "Dates" ] = exchange_rates[ "Dates" ].apply( lambda x: pd.to_datetime(
```

1.1.7 Making Dataset Uniform

In the cell below: - I use the previous data set to convert each currency that isn't in USD to USD.

```
In [74]: def change_to_USD(x):
             date = x[ 3 ]
             if x[ 7 ] != "$" and x[ 7 ] != "nan":
                 try:
                     cur_rate = exchange_rates[ x[ 7 ] ][ exchange_rates[ "Dates" ] == date ].va
                     x[ 4 ] = int( x[ 4 ] ) / float( cur_rate )
                 except:
                     x[ 4 ] = np.nan
             elif x[ 8 ] != "$" and x[ 8 ] != "nan":
                 try:
                     cur_rate = exchange_rates[ x[ 8 ] ][ exchange_rates[ "Dates" ] == date ].va
                     x[ 5 ] = int( x[ 5 ] ) / float( cur_rate )
                 except:
                     x[ 5 ] = np.nan
             return x
         df1 = df1.apply( change_to_USD, axis = 1 )
```

1.1.8 Data Cleaned and Uniform

In the cell below: - I set my original dataset to the dataset that I have been manipulating so they are both the same

```
In [75]: del df1[ 7 ]
         del df1[ 8 ]
         df = df1
         df.columns = columns
         df.sample( 5 )
```

```
Out[75]:                                                 Name Rating  Score    Release  \
         633       Marina Abramovic: The Artist Is Present     NR   95.0 2012-07-05
         446                           Jafar Panahi's Taxi     NR   96.0 2015-04-15
         24   Harry Potter and the Deathly Hallows - Part 2   PG13   96.0 2011-07-15
         527                                     Suspiria      R   93.0 2018-01-01
         758                                   Food Inc.     PG   95.0 2009-07-31

                Budget         Open        Gross
         633       NaN      11041.0      86217.0
         446       NaN        166.0     321642.0
         24   125000000.0  169189427.0  381011219.0
         527       NaN          NaN          NaN
         758       NaN      40744.0    4417124.0
```

1.1.9 Units

For the data set above: - Dates have been uniformly written, using pands.to_datetime - All currency is in USD

Note: It may be misleading that some entries in the opening weekends have a much larger value than the gross profit. This is because the opening weekends column describes all the money spent on that movie during the opening weekend. This is different from the gross, because the gross describes the profit after everyone has been paid.

```
In [76]: df.to_csv( "Cleaned_Data.csv" )
```

2 Initial Feature Engineering

In the cell below I create three new columns that can be useful later, when trying to detect trends in the data. - The first new column is a ratio of the budget of the movie to the money spent opening weekend. The second column is a ratio of the budget to the gross profit the movie made. The last new column is the ratio of the money spent on opening weekend to the gross profit made. - I added these columns because it would make it easier to see if there was any type of correlation between factors contributing to ow much a movie made. I also fill in missing values with the median values

```
In [77]: owk_median = df[ "Open" ].median()
         gro_median = df[ "Gross" ].median()
         bud_median = df[ "Budget" ].median()
         df[ "Open" ] = df[ "Open" ].fillna( value = owk_median )
         df[ "Gross" ] = df[ "Gross" ].fillna( value = gro_median )
         df[ "Budget" ] = df[ "Budget" ].fillna( value = bud_median )
         df[ "Budget.Open" ]  = df[ "Budget" ] / df[ "Open" ]
         df[ "Open.Gross" ]   =   df[ "Open" ] / df[ "Gross" ]
         df[ "Budget.Gross" ] = df[ "Budget" ] / df[ "Gross" ]

In [78]: df = df.dropna()
         def decade( x ):
             year = x.year
```

```
        return ( ( year % 1900 ) // 10 ) * 10 + 1900
    df[ "Decade" ] = df[ "Release" ].apply( decade ) # Calculating the decade for the relea
    df.sample(5)
```

Out[78]:

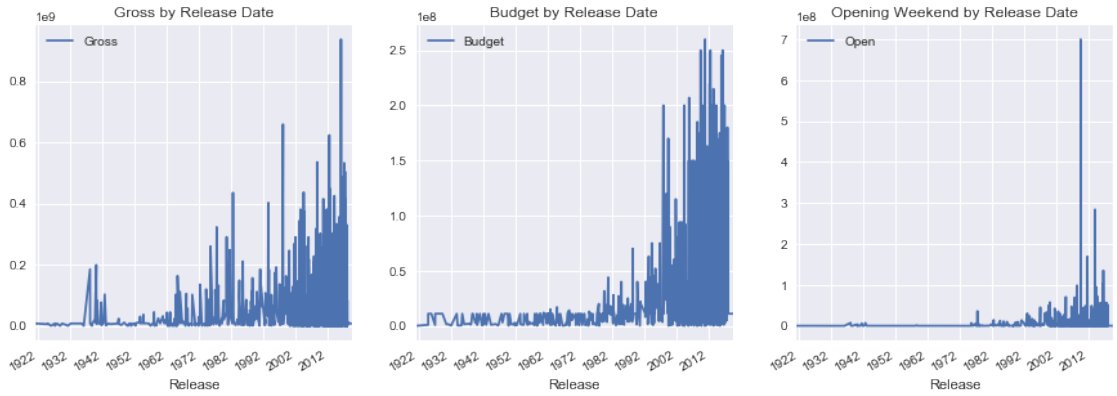
	Name	Rating	Score	Release	Budget	Open	\
795	Hoosiers	PG	88.0	1987-02-27	6.000000e+06	220068.0	
524	Shaun of the Dead	R	92.0	2004-09-24	2.218402e+06	1603410.0	
152	The Little Prince	PG	93.0	2016-08-05	8.120000e+07	3113945.0	
530	Eraserhead	R	91.0	1978-02-03	2.000000e+04	151940.5	
632	The Wrecking Crew	PG	94.0	1968-12-25	1.100000e+07	151940.5	

	Gross	Budget.Open	Open.Gross	Budget.Gross	Decade
795	28607524.0	27.264300	0.007693	0.209735	1980
524	13542874.0	1.383552	0.118395	0.163806	2000
152	1339152.0	26.076247	2.325311	60.635387	2010
530	7000000.0	0.131630	0.021706	0.002857	1970
632	7707563.0	72.396761	0.019713	1.427170	1960

```
In [79]: df.to_csv( "Engineered_Data.csv" )
```

3 Visualizing Data

```
In [80]: plt.figure( figsize = ( 15, 5 ) )
         ax1, ax2, ax3 = plt.subplot( 131 ), plt.subplot( 132 ), plt.subplot( 133 )
         df.plot( y = "Gross",  x = "Release", ax = ax1, title = "Gross by Release Date" )
         df.plot( y = "Budget", x = "Release", ax = ax2, title = "Budget by Release Date" )
         df.plot( y = "Open",   x = "Release", ax = ax3, title = "Opening Weekend by Release Dat
         plt.show()
```

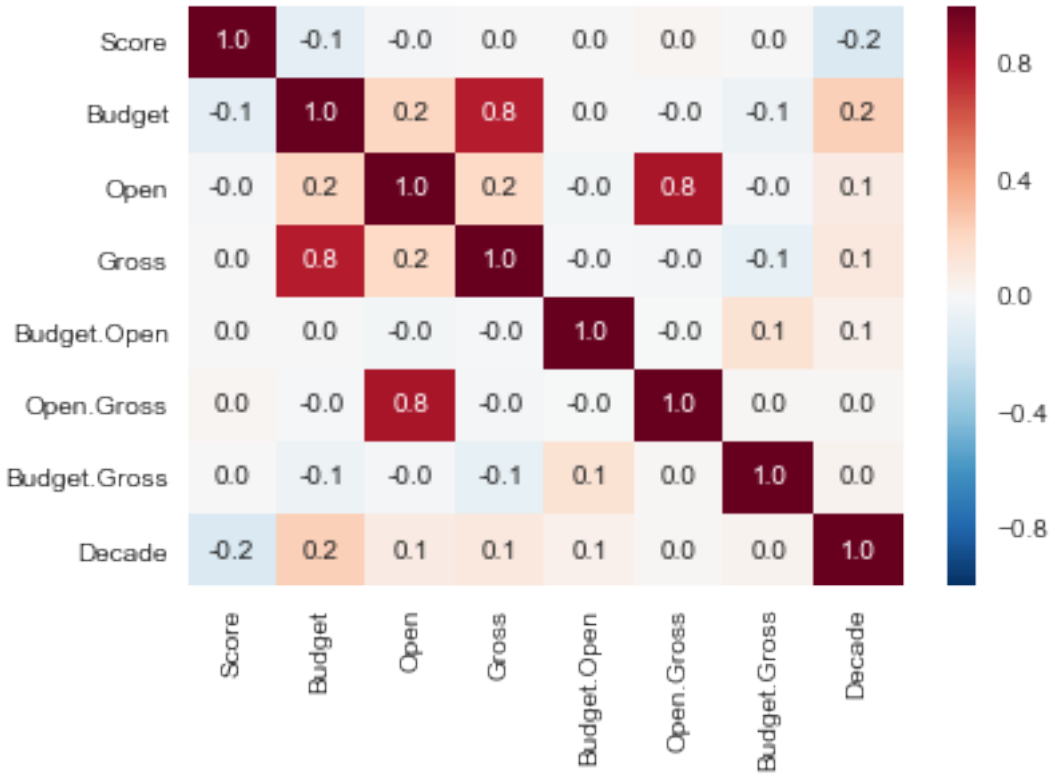


From these graphs it's noticable that the budget, gross profit, and opening weeking for movies has exponentially increased over the past century.

```
In [103]: seaborn.pairplot(df[ [ "Budget", "Open", "Gross", "Score" ,"Decade", "Rating" ] ], hue
          plt.show()
```

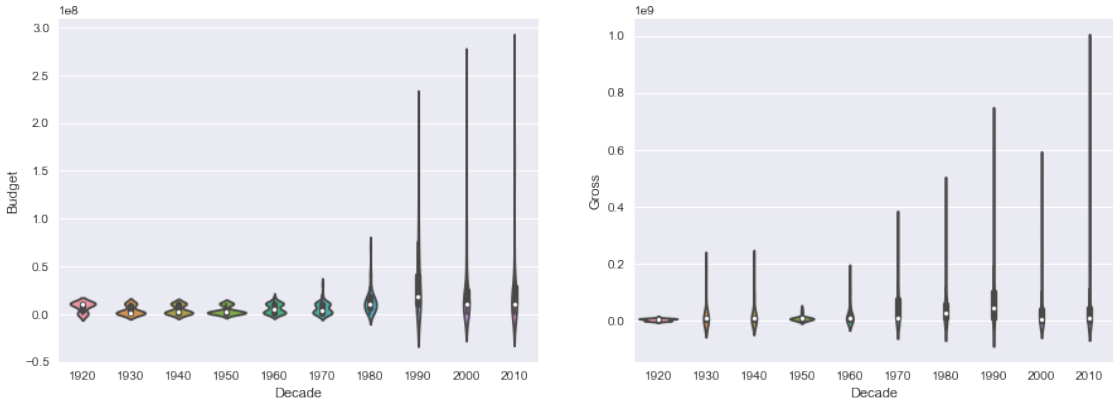
Observing the scatter matrix above, the budget and gross profit of a movie have a strong positive correlation, while gross profit and opening weekend, and opening weekend and budget both have weaker correlations.

```
In [98]: seaborn.heatmap( df.corr(), annot = True, fmt = ".1f" )
         plt.show()
```



This heatplot shows that there is a stong correlation between the budget and gross, with a correlation coefficint of .7 (confirming what the scatterplot demonstrated), also there is a slight positive correlation between the release decade and the budget for the movie. There is also a correlation between the opening weekend and gross, with a coefficient of .3. Interestingly, none of the ratios are in any way correlated to each other.

```
In [83]: plt.figure( figsize = ( 15, 5 ) )
         ax1, ax2 = plt.subplot( 121 ), plt.subplot( 122 )
         seaborn.violinplot( y = "Budget", x = "Decade", data = df, ax = ax1 )
         seaborn.violinplot( y = "Gross",  x = "Decade", data = df, ax = ax2 )
         plt.show()
```



The first plot shows the distribution of the budget for each decade using the width of each "violin". This violin plot above demonstrates a much larger variance in the budgets for movies in later decades than in previous decades.The second violin plot for the gross profit of movies versus the decade, shows that the gross profit for movies is significantly more variant than the budget for the movies.

```
In [86]: plt.figure( figsize = ( 15, 5 ) )
         ax1, ax2 = plt.subplot( 121 ), plt.subplot( 122 )
         seaborn.violinplot( y = "Gross",  x = "Rating", data = df, ax = ax1 )
         seaborn.violinplot( y = "Score",  x = "Rating", data = df, ax = ax2 )
         plt.show()
```