AIOverlordSupporters / Notebook.ipynb

jumholtz Final Updates

History

1 contributor

1277 lines (1277 sloc) | 126 KB

Project Overview

Group Members: Maya Sandlin Jake Umholtz Robert Golden Daniel Robles Maanik Gupta

This notebook contains our project goals, data, methods, and results

Business Understanding

Computing Vision is a relative newcomer to the film production market, having little to no experience producing movies; for this reason, Computing Vision selected Deloitte to advise them on the creation of an initial, data-driven film production strategy.

Our measure of success was domestic Box Office revenue. We chose this metric because box office revenue is the first way film studios generate a cash flow. As a new movie studio, it is important for Computing Vision to start generating a cash flow as quickly and as largely as possible. Also, the production of films is an expensive endeavor, a successful box office performance can help recover the initial investments of creating a film. Box office revenue is also a good indicator of public sentiment as it indirectly measures how many people are physically going to watch the film in theaters. This serves as baseline check for future questioning regarding at home and digital release.

With Box Office revenue as our primary indicator of success, we used data to answer the following question; what components of a film creates a box office hit?

We quickly identified runtime, genre, and MPAA rating as foundational components for creating a box office hit. We identified these three components as they are the key drivers to reaching a large audience. If Computing Vision's films can capture a large audience, they have a higher probability of creating a box office hit, i.e. attract more people to pay to watch the film.

Data Understanding

We focused our analysis on the Rotten Tomatoes Movie Information dataset, known as the rt_movie_info spreadsheet as it contained the necessary box office, genre, and movie length information.

We also used the The Numbers movie budget dataset to calculate the median movie budget as well as how many films had budgets at and above $100 million. This information supported our claim on why focusing on Box Office revenue is vital for Computing Vision.

Data Preparation and Analysis

For each recommendation - genre, runtime, MPAA rating - we chose to drop the null values because the data set is large enough at over 1500 records to handle the loss of data. Also, much of the data set is highly varied, so using the mean or median would not be a fully accurate representation.

Rationale: Import the necessary python libraries to perform analysis

In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

Importing data and naming variables

In [4]:

```
movie_gross = pd.read_csv('zippedData/bom.movie_gross.csv')
rt_movie_info = pd.read_csv('zippedData/rt.movie_info.csv')
rt_review = pd.read_csv('zippedData/rt.reviews.csv', encoding='unicode_escape')
tn_movie_budgets = pd.read_csv('zippedData/tn.movie_budgets.csv')
tmdb_movies = pd.read_csv('zippedData/tmdb.movies.csv', index_col=0)
```

Analysis for the median movie budget and for movies that had a budget at or above $100 million. This information helps inform our business design on using Box Office Revenue as our measure of success. With budgets being so large, it is important for Computer Vision to recover their investment (budget) as quickly as possible via the Box Office. By calculating the median we are able to generate an insight on how much they should seek to recover in the box office. By uncovering the number of films with budgets at or 100 million USD that helps us paint a more accurate picture of why Box Office Revenue is important.

In [5]:

In [5]:

```
# checking for null values
print(tn_movie_budgets.isnull().sum())
```

```
id                  0
release_date        0
movie               0
production_budget   0
domestic_gross      0
worldwide_gross     0
dtype: int64
```

Data cleaning for movie budgets and calculating the median budget and number of films with production budget over $100 million

In [6]:

```
# turning the budget into a string so it is easy to clean
tn_movie_budgets['production_budget'] = tn_movie_budgets['production_budget'].astype(str)

# removing commas, dollar signs, and white space
# turning the production budget back into an integer so calculations can be performed
tn_movie_budgets['production_budget'] = tn_movie_budgets['production_budget'].str.replace(',', '')
tn_movie_budgets['production_budget'] = tn_movie_budgets['production_budget'].str.replace('$', '')
tn_movie_budgets['production_budget'] = tn_movie_budgets['production_budget'].str.replace(' ', '')
tn_movie_budgets['production_budget'] = tn_movie_budgets['production_budget'].astype(int)

# calculating median
median_budget = tn_movie_budgets['production_budget'].median()

# counting the number of films with a production budget over $100 million
mill_budget = tn_movie_budgets[tn_movie_budgets['production_budget'] >= 100000000].count()

print(f'Median Production Budget: $ {median_budget}')
```

```
Median Production Budget: $ 17000000.0
```

In [7]:

```
print(f'Number of films with a budget that is >= $100 million: {mill_budget}')
```

```
Number of films with a budget that is >= $100 million: id                   406
release_date        406
movie               406
production_budget   406
domestic_gross      406
worldwide_gross     406
dtype: int64
```

Genre

Data cleaning for genres

In [10]:

```
# segregating genre and box office from the dataset
movie_genre = rt_movie_info[['genre','box_office']]
# cleaning the segregated variables by dropping null values
clean_genre = movie_genre.dropna()
genres_sorted = movie_genre['genre'].sort_values().dropna()
genres_sorted.value_counts().head (30)
#using this data, the main six categories include: drama, comedy, horror, action, romance, and classics

#determining the average and median revenue by genre

#drama
drama = clean_genre[clean_genre['genre'].str.contains('Drama')]
drama_revenue = drama ['box_office']
drama_revenue_clean = drama_revenue.dropna()
drama_avg_revenue = np.mean(drama_revenue_clean)
drama_med = np.median(drama_revenue_clean)

#comedy
comedy = clean_genre[clean_genre['genre'].str.contains('Comedy')]
comedy_revenue = comedy ['box_office']
comedy_revenue_clean = comedy_revenue.dropna()
comedy_avg_revenue = np.mean(comedy_revenue_clean)
comedy_med = np.median(comedy_revenue_clean)

#action
action = clean_genre[clean_genre['genre'].str.contains('Action')]
action_revenue = action ['box_office']
action_revenue_clean = action_revenue.dropna()
action_avg_revenue = np.mean(action_revenue_clean)
action_med = np.median(action_revenue_clean)

#horror
horror = clean_genre[clean_genre['genre'].str.contains('Horror')]
horror_revenue = horror ['box_office']
horror_revenue_clean = horror_revenue.dropna()
horror_avg_revenue = np.mean(horror_revenue_clean)
horror_med = np.median(horror_revenue_clean)
```

```
#romance
romance = clean_genre[clean_genre['genre'].str.contains('Romance')]
romance_revenue = horror ['box_office']
romance_revenue_clean = romance_revenue.dropna()
romance_avg_revenue = np.mean(romance_revenue_clean)
romance_med = np.median(romance_revenue_clean)

#classics
classics = clean_genre[clean_genre['genre'].str.contains('Classics')]
classic_revenue = classics ['box_office']
classic_revenue_clean = classic_revenue.dropna()
classic_avg_revenue = np.mean(classic_revenue_clean)
classic_med = np.median(classic_revenue_clean)
```

By calculating the averages and medians for each of the top genres, this shows us the best genre of films in regards to Box Office Revenue. From these findings, we are able to conclude that Action films produce the most Box Office Revenue.

MPAA Rating

Data cleaning for MPAA ratings

In [11]:
```
#made copy of df to drop Nan values and generate insights
movie_info = rt_movie_info.copy()
```

In [12]:
```
#drop records with nan values
movie_info = movie_info.dropna()
```

In [13]:
```
# turing box office into a string for cleaning
movie_info['box_office'] = movie_info['box_office'].astype(str)
```

In [14]:
```
#remove commas
movie_info['box_office']=movie_info['box_office'].str.replace(',','')
```

In [15]:
```
# making it so that we can sort and graph by the budget values
movie_info['box_office']=movie_info['box_office'].astype(float)
```

In [16]:

```
# creating variables for each rating, narrowing down the table to that respective rating
R = movie_info[movie_info['rating']=='R']
PG = movie_info[movie_info['rating']=='PG']
PG13 = movie_info[movie_info['rating']=='PG-13']
G = movie_info[movie_info['rating']=='G']
NC17 = movie_info[movie_info['rating']=='NC17']
NR = movie_info[movie_info['rating']=='NR']
```

In [17]:

```
#new variables for the mean and median box office revenue
r=R['box_office'].mean()
rmed=R['box_office'].median()
pg=PG['box_office'].mean()
pgmed=PG['box_office'].median()
pg13=PG13['box_office'].mean()
pg13med=PG13['box_office'].median()
g=G['box_office'].mean()
gmed=G['box_office'].median()
nc17=NC17['box_office'].mean()
nc17med=NC17['box_office'].median()
nr=NR['box_office'].mean()
nrmed=NR['box_office'].median()
```

Creating a dictionary for easier graphing

In [18]:

```
#this dictionary containing the mean variables will help with graphing
ratings_dict = {'R':r,'PG':pg,'PG-13':pg13,'G':g,'NC-17':nc17,'NR':nr}
```

In [19]:

```
#same as previous dictionary but for median variables
ratings_dict_med = {'R':rmed,'PG':pgmed,'PG-13':pg13med,'G':gmed,'NC-17':nc17med,'NR':nrmed}
```

By calculating averages and medians for each of the MPAA ratings, this provides us with the best MPAA rating in regards to Box Office Revenue. From these findings, we are able to conclude that films rated PG produce the most Box Office Revenue.

Runtime

Data cleaning for runtime

In [20]:

```
# renaming variables for sake of clarity
rt_movie_info_runtime = rt_movie_info
```

In [21]:

```
# dropping null values
rt_movie_info_runtime = rt_movie_info.dropna()

box_office = []
for v in rt_movie_info_runtime['box_office'].dropna():
    box_office.append(box_office)
```

In [22]:

```
runtime_raw = [] #for freshly-extracted runtimes
runtime_clean = [] #for runtime w/o whitespace

for val in rt_movie_info_runtime['runtime'].str[:3]: #extract first 3 characters, movies under 100 min will ha
    runtime_raw.append(val)
for val in runtime_raw: #strip whitespace
    runtime_strip = val.replace(' ', '')
    runtime_clean.append(runtime_strip)

#create new column 'runtime_clean'
rt_movie_info_runtime['runtime_clean'] = runtime_clean
```

```
<ipython-input-22-2b1f70847de7>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  rt_movie_info_runtime['runtime_clean'] = runtime_clean
```

In [23]:

```
# creating new column for future analysis
rt_movie_info_runtime['box_office_clean'] = rt_movie_info_runtime['box_office']

# viewing dtypes in dataframe
rt_movie_info_runtime.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 235 entries, 1 to 1545
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   id                235 non-null    int64  
 1   synopsis          235 non-null    object 
 2   rating            235 non-null    object 
 3   genre             235 non-null    object 
 4   director          235 non-null    object 
 5   writer            235 non-null    object 
 6   theater_date      235 non-null    object 
 7   dvd_date          235 non-null    object 
 8   currency          235 non-null    object 
 9   box_office        235 non-null    float64
 10  runtime           235 non-null    object 
 11  studio            235 non-null    object 
 12  runtime_clean     235 non-null    object 
 13  box_office_clean  235 non-null    float64
dtypes: float64(2), int64(1), object(11)
memory usage: 27.5+ KB
```

```
<ipython-input-23-67149c2021ed>:2: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  rt_movie_info_runtime['box_office_clean'] = rt_movie_info_runtime['box_office']
```

In [24]:

```
# changing objects to strings for further cleaning
rt_movie_info_runtime = rt_movie_info_runtime.astype(str)

# removing the comma from box office for easier future analysis
rt_movie_info_runtime['box_office_clean'].map(lambda x: float(x.replace(',', '')))
```

Out[24]:

```
1         600000.0
6       41032915.0
7         224114.0
15       1039869.0
18      20518224.0
           ...    
1530    72700000.0
1537     1320005.0
1541    25335935.0
1542     1416189.0
1545       59371.0
Name: box_office_clean, Length: 235, dtype: float64
```

In [25]:

```
# changing box office back into a float for analysis
```

```
# changing box office back into a float for analysis
rt_movie_info_runtime['box_office_clean'] = rt_movie_info_runtime['box_office_clean'].astype(float)
```

In [26]:

```
#create new dataframe
df2 = rt_movie_info_runtime[['runtime_clean','box_office_clean']].copy()

#ensure runtime valuse are recognized as integers
df2['runtime_clean'] = df2['runtime_clean'].astype(int)

#ensure box office slaes values are recognized as integers
df2['box_office_clean'] = df2['box_office_clean'].astype(int)
```

In [27]:

```
#Descriptive Stats for Runtime
max1 = df2['runtime_clean'].max()
Q1_3 = df2['runtime_clean'].quantile(q=0.75)
mean1 = df2['runtime_clean'].mean()
med1 = df2['runtime_clean'].median()
std1 = df2['runtime_clean'].std()
Q1_1 = df2['runtime_clean'].quantile(q=0.25)
min1 = df2['runtime_clean'].min()

print('Descriptive Statistics for Runtime')
print('Mean runtime:', mean1)
print('')
print('Maximum runtime:', max1)
print('75th Percentile:', Q1_3)
print('Median runtime:', med1)
print('25th Percentile:', Q1_1)
print('Minimum runtime:', min1)
print('Interquartile Range:', Q1_3-Q1_1)
print('Std. Dev. of runtime:', std1)

print('99.7% of observations should lay between:', mean1-(std1*2), '-',mean1+(std1*2))

print('') #for readability

#Descriptive Stats for Box Office Sales
mean2 = df2['box_office_clean'].mean()
max2 = df2['box_office_clean'].max()
Q2_3 = df2['box_office_clean'].quantile(q=0.75)
med2 = df2['box_office_clean'].median()
Q2_1 = df2['box_office_clean'].quantile(q=0.25)
min2 = df2['box_office_clean'].min()
std2 = df2['box_office_clean'].std()
```

```
std2 = df2['box_office_clean'].std()

print('Descriptive Statistics for Box Office Sales')
print('Mean box office sales:', mean2)
print('')
print('Maximum sales:', max2)
print('75th Percentile:', Q2_3)
print('Median sales:', med2)
print('25th Percentile:', Q2_1)
print('Minimum sales:', min2)
print('Std. Dev of Sales:', std2)
print('Interquartile Range:', Q2_3-Q2_1)
print('99.7% of observations should lay between:', mean2-(std2*2), '-', mean2+(std2*2222))

print('') #for readability
print('Correlation Coefficient')
print(df2.corr()) #calculate Pearson correlation coefficient for variables in df2

#generate a normal distribution where mean1 is avg, sd1 is std dev, and n=235
d1 = np.random.normal(mean1, std1, 235)
#generate a normal distribution where mean2 is avg, sd2 is std dev, and n=235
d2 = np.random.normal(mean2, std2, 235)

df2more = df2[df2['runtime_clean'] > 105]
df2less = df2[df2['runtime_clean'] < 105]
print('')
print('Less - Runtime Mean:', df2less['runtime_clean'].mean(),
      'Box Office Sales Mean:', df2less['box_office_clean'].mean(),
      'n:', len(df2less))
print('More - Runtime Mean:', df2more['runtime_clean'].mean(),
      'Box Office Sales Mean:', df2more['box_office_clean'].mean(),
      'n:', len(df2more))
#df2['runtime_clean'].plot.box(grid='True')
#df2['box_office_clean'].plot.box(grid='True')
```

```
Descriptive Statistics for Runtime
Mean runtime: 106.66382978723404

Maximum runtime: 188
75th Percentile: 117.0
Median runtime: 105.0
25th Percentile: 93.0
Minimum runtime: 67
Interquartile Range: 24.0
Std. Dev. of runtime: 18.14712458129922
99.7% of observations should lay between: 70.3695806246356 - 142.95807894983247
```

```
Descriptive Statistics for Box Office Sales
Mean box office sales: 41958400.02127659

Maximum sales: 368000000
75th Percentile: 52649522.5
Median sales: 15536310.0
25th Percentile: 2302444.5
Minimum sales: 363
Std. Dev of Sales: 62630155.518367976
Interquartile Range: 50347078.0
99.7% of observations should lay between: -83301911.01545936 - 139206163961.8349

Correlation Coefficient
                  runtime_clean  box_office_clean
runtime_clean          1.000000          0.312157
box_office_clean       0.312157          1.000000

Less - Runtime Mean: 92.43103448275862 Box Office Sales Mean: 28014808.870689657 n: 116
More - Runtime Mean: 121.21929824561404 Box Office Sales Mean: 56785095.37719298 n: 114
```

By calculating the correlation between runtime and Box Office Revenue, we are able to uncover a potential relationship between the two variables. However, as the Pearson Coefficent is not very strong, we cannot conclude a strong relationship. This however, is still a useful business insight as runtime is a very important and costly decision in terms of budgeting and capturing audiences.

Visualization

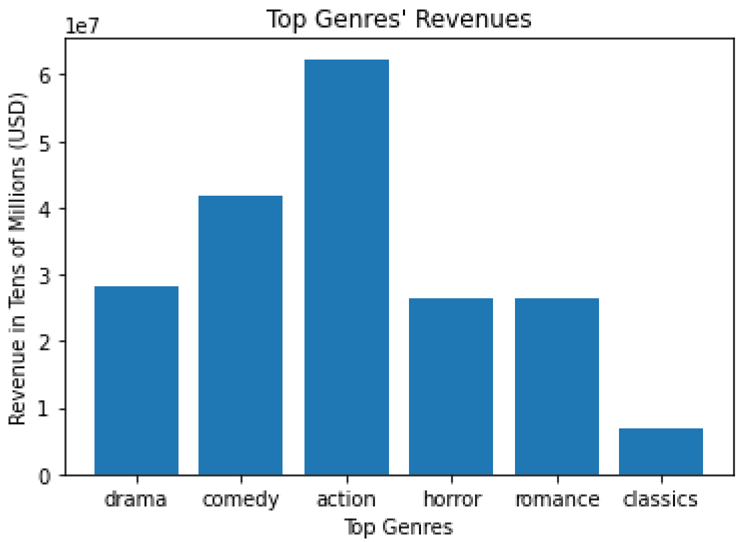
Genre

Creating a bar graphs to compare genres (genres are a categorical variable)

In [28]:

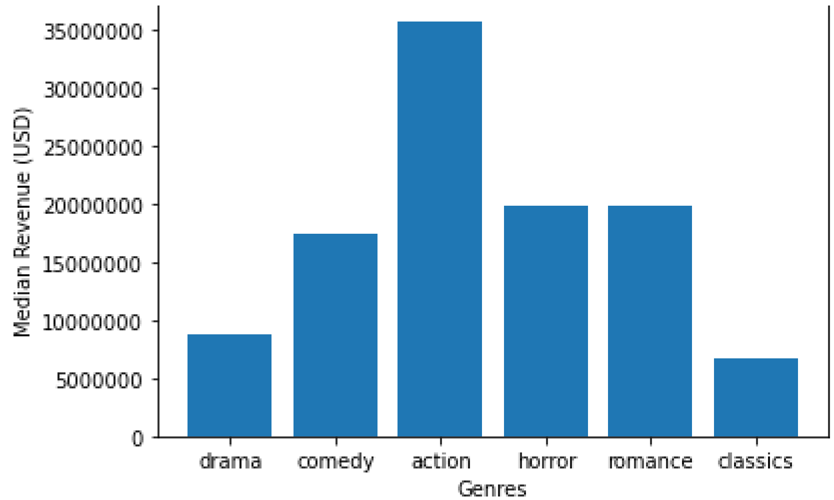
```
top_genre={"drama":drama_avg_revenue, "comedy":comedy_avg_revenue, "action":action_avg_revenue,
           "horror": horror_avg_revenue, "romance":romance_avg_revenue, "classics": classic_avg_revenue}

#creating the graph to compare the top genres
import matplotlib.pyplot as plt
%matplotlib inline
```

```
fig, ax=plt.subplots()
# plt.ticklabel_format(style='plain')
ax.bar(top_genre.keys(),top_genre.values())



ax.set_title("Top Genres' Revenues")
ax.set_xlabel("Top Genres")
ax.set_ylabel("Revenue in Tens of Millions (USD)");
```



In [29]:

```
top_genre_med = {"drama":drama_med, "comedy":comedy_med, "action":action_med,
           "horror": horror_med, "romance":romance_med, "classics": classic_med}

fig, ax = plt.subplots()
plt.ticklabel_format(style = 'plain')
ax.bar(top_genre_med.keys(),top_genre_med.values() )





ax.set_title("Median Box Office Revenue by Genre")
ax.set_xlabel("Genres")
ax.set_ylabel("Median Revenue (USD)");
```

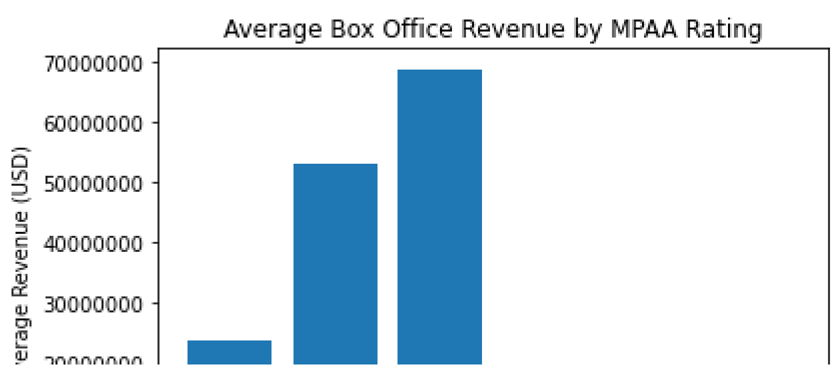


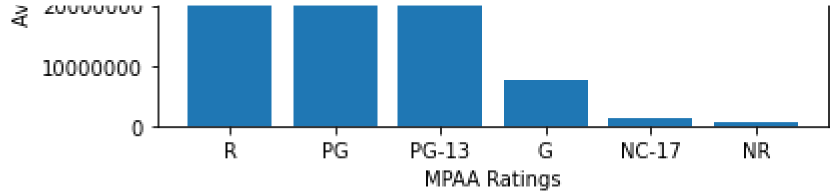
MPAA Rating

Creating bar graphs to compare MPAA ratings (MPAA ratings are categorical variables)

In [30]:

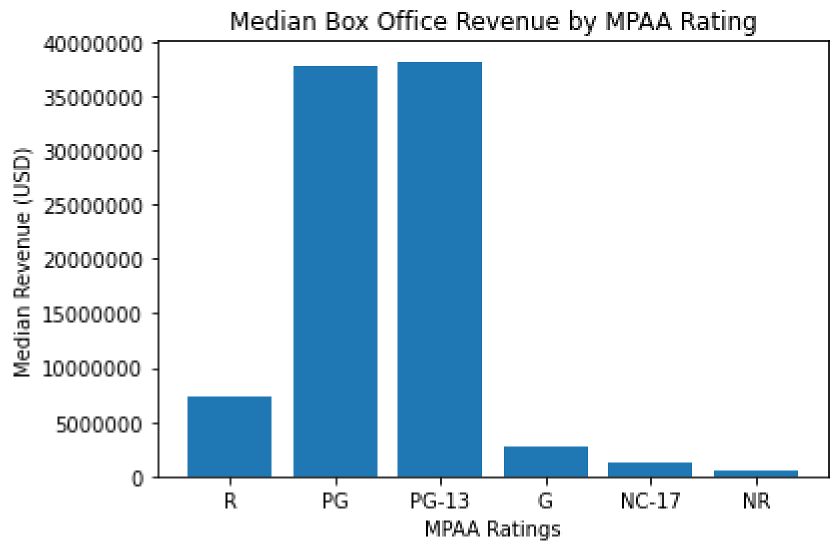
```
#Mean box office revenue by rating
fig, ax=plt.subplots()
plt.ticklabel_format(style='plain')
ax.bar(ratings_dict.keys(),ratings_dict.values() )



ax.set_title("Average Box Office Revenue by MPAA Rating")
ax.set_xlabel("MPAA Ratings")
ax.set_ylabel(" Average Revenue (USD)");
```





In [31]:

```
#median box office revenue by rating
fig, ax=plt.subplots()
plt.ticklabel_format(style='plain')
ax.bar(ratings_dict_med.keys(),ratings_dict_med.values() )



ax.set_title("Median Box Office Revenue by MPAA Rating")
ax.set_xlabel("MPAA Ratings")
ax.set_ylabel("Median Revenue (USD)");
```



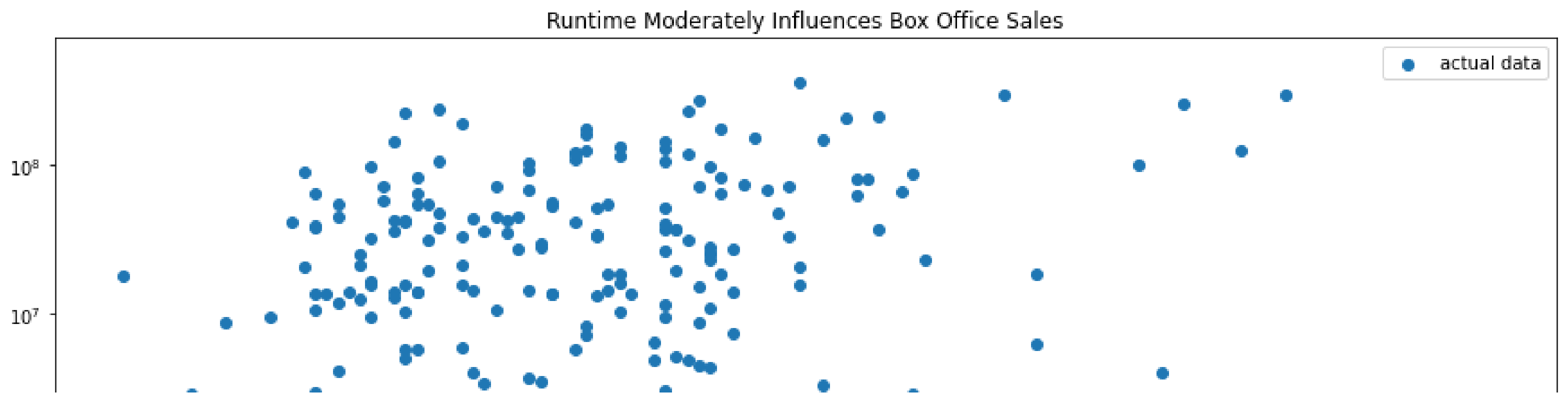
Runtime

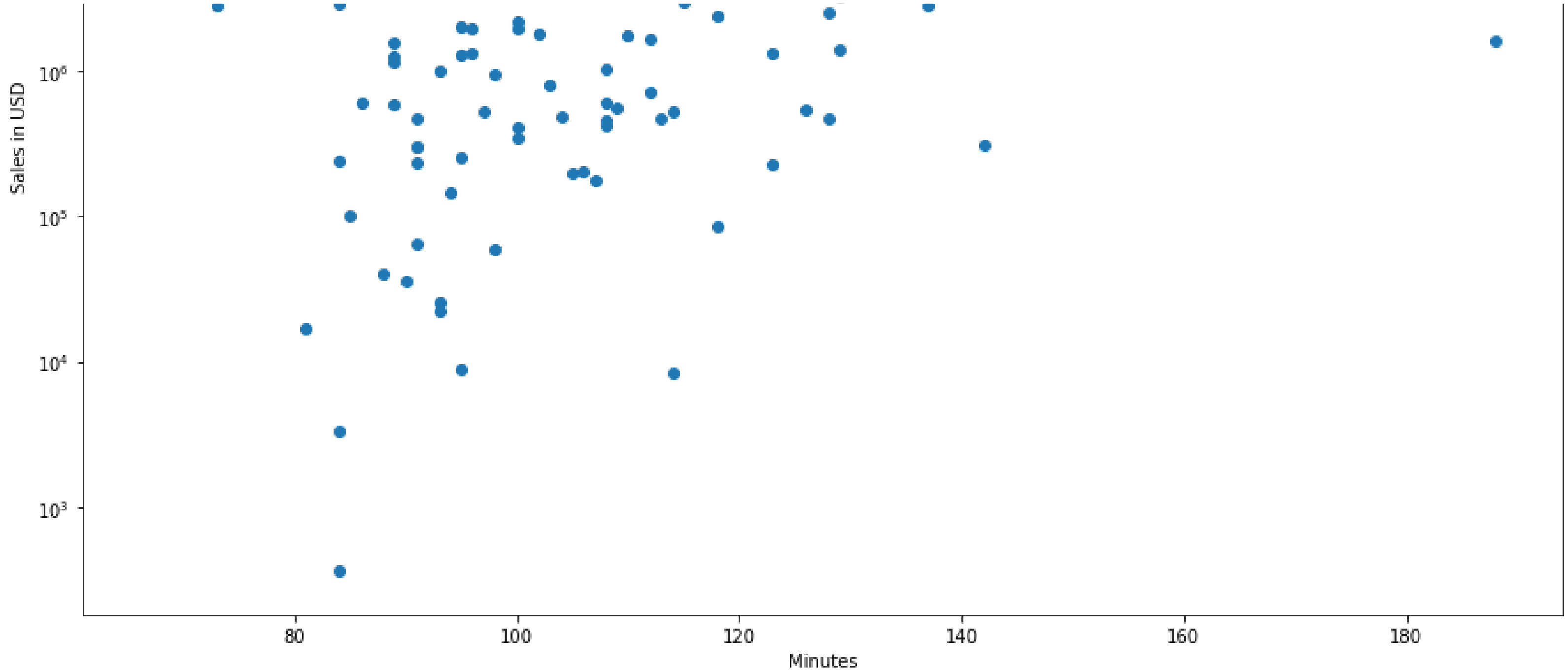
Creating a scatter plot to illustrate the relationship between Box Office Revenue and runtime

In [32]:

```
fig, ax = plt.subplots(figsize=(15,10)) #Generate Scatter plot
```

```
ax.scatter(df2['runtime_clean'], df2['box_office_clean'], label="actual data")



x_bounds = [min(df2['runtime_clean']), max(df2['runtime_clean'])]
y_bounds = [min(df2['box_office_clean']), max(df2['box_office_clean'])]


x_bounds_log = np.log2(x_bounds)
y_bounds_log = np.log(y_bounds)





ax.set_title("Runtime Moderately Influences Box Office Sales")
ax.set_xlabel("Minutes")
ax.set_ylabel("Sales in USD")
plt.yscale("symlog")





ax.legend();
```





Statistical Communication

Conducted a one-tailed t-test to test the relationship between the action genre and box office performance

Null Hypothesis: The action genre has no effect on box office revenue Alternative Hypothesis: Action films perform better in terms of box office revenue

In [33]:
```
# defining variables for t-test
pop=movie_info['box_office']
act=action['box_office']
alpha = .05

stats.norm.ppf(alpha), stats.norm.ppf(1-alpha)
```

Out[33]:
```
(-1.6448536269514729, 1.6448536269514722)
```

In [34]:
```
stats.ttest_ind(act,pop)
```