- Student name: **Daniel M. Smith**
- Student pace: **Full Time Online**
- Scheduled project review date/time: **May 4, 2021 11 AM**
- Instructor name: **Project Reviewer: Claude Fried**
  Cohort Lead: Abhineet Kulkarni

- Blog post URL: https://danielmsmith1.medium.com/pivot-vs-pivottable-vs-groupby-2d8723beb782

Daniel M. Smith
Movie Studio Project Phase 1 Flatiron School

Business Understanding

Microsoft sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. You are charged with exploring what types of films are currently doing the best at the box office. You must then translate those findings into actionable insights that the head of Microsoft's new movie studio can use to help decide what type of films to create.

Data Mining

Initial data was provided and can be downloaded here.

Data Cleaning

In this workbook we:

1. Load the data into pandas DataFrames
2. Inspect and observe the DataFrames
3. Clean and convert the data into appropriate types

Initial Reading of data files

In [3]:
```
import pandas as pd
import numpy as nm
from os import listdir
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker

path = '../zippedData/'
```

In [4]:
```
#helper to list all csv or type files in a dir
def find_csv_filenames( path_to_dir, suffix=".csv"):
    filenames = listdir(path_to_dir)
    return [ filename for filename in filenames if filename.endswith( suffix ) ]

#creates dataframes for type specified
def create_dfs(filelist, suffix=".csv" ):
    #Read all the files and store in a dataFrame
    # the data Frames for each file will be listed in a dict
    # where key is the name and value is the df
    dict_csv_files = {}

    for filename in csvfiles:
        filename_cleaned = filename.replace(".csv", "").replace(".","_")#cleaning
        filename_df = pd.read_csv(path + filename, index_col = 0, encoding='utf8')
        dict_csv_files[filename_cleaned] = filename_df
    return dict_csv_files
```

In [5]:
```
#Create csvfiles, tsvfiles and call createdfs dict
csvfiles = find_csv_filenames(path)
tsvfiles = find_csv_filenames(path,'.tsv')
dict_dfs = create_dfs(csvfiles, suffix=".csv" )
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-5-e330f7f8e661> in <module>
      1 #Create csvfiles, tsvfiles and call createdfs dict
----> 2 csvfiles = find_csv_filenames(path)
      3 tsvfiles = find_csv_filenames(path,'.tsv')
      4 dict_dfs = create_dfs(csvfiles, suffix=".csv" )

<ipython-input-4-90ae2e2ee2f8> in find_csv_filenames(path_to_dir, suffix)
      1 #helper to list all csv or type files in a dir
      2 def find_csv_filenames( path_to_dir, suffix=".csv"):
----> 3     filenames = listdir(path_to_dir)
      4     return [ filename for filename in filenames if filename.endswith( suffix ) ]
      5
```

```
FileNotFoundError: [WinError 3] The system cannot find the path specified: '../zippedData/'
```

In [ ]:
```
dict_dfs.keys()
```

In [ ]:
```
##Create Working DataFrames
df_bom = dict_dfs['bom_movie_gross']
df_imdb_name = dict_dfs['imdb_name_basics']
df_imdb_akas = dict_dfs['imdb_title_akas']
df_imdbbasics = dict_dfs['imdb_title_basics']
df_imdb_crew = dict_dfs['imdb_title_crew']
df_imdb_principals = dict_dfs['imdb_title_principals']
df_imdb_ratings = dict_dfs['imdb_title_ratings']
df_tmb = dict_dfs['tmdb_movies']
df_tn_movie_budget = dict_dfs['tn_movie_budgets']
#Excluding rott.tsvfiles  df_rott_info = dict_dfs(' pd.read_csv('zippedData/rt.movie_info.tsv', sep='\t')
#df_rott_rev = dict_dfs(' pd.read_csv('zippedData/rt.reviews.tsv',encoding= 'unicode_escape', sep='\t')
```

Inspect DataFrames

Inspecting the dfs, Noting observations about the data, describing the data types.

We have 9 DataFrames from the 9 files:

df_bom

df_imdb_name

df_imdb_akas

df_imdbbasics

df_imdb_crew

df_imdb_principals

df_imdb_ratings

df_tmb

df_tn_movie_budget

Box Office Mojo-df_bom

In [ ]:
```
df_bom.info()
```

In [ ]:
```
df_bom.head(3)
```

Observation df_bom (Box office Mojo)

It looks as if the data is for 2010 to 2018 movies of domestic and foreign gross in dollars.
Foreign gross needs to be converted to int datatype.

IMDB name basics-df_imdb_name

In [ ]:
```
df_imdb_name.info()
```

In [ ]:
```
df_imdb_name.head(3)
```

Observation df_imdb_name

It looks as if the data is a name and profession and known for these movie titles.
The birth year and death year can be dropped as there is a high percentage of data missing from those columns.

IMDB akas-df_imdb_akas

In [ ]:
```
df_imdb_akas.info()
```

In [ ]:
```
df_imdb_akas.head(3)
```

Observation df_imdb_akas

This data looks to be aka movie names in non domestic markets. There are lots of NaNs. Should we just look at isoriginal title as 1? Where does the title_id link to? Is ok to leave NaNs alone?

IMDBTitle basics-df_imdbbasics

In [ ]:
```
df_imdbbasics.info()
```

In [ ]:
```
df_imdbbasics.tail(3)
```

In [ ]:
```

```

Observation df_df_imdbbasics

This data has primary title and original title and year and genre of the movie. Many missing runtime minutes can set to 90 mins? Main point here is the genre and year and title.

IMDB Crew-df_imfb_crew

In [ ]:
```
df_imdb_crew.info()
```

In [ ]:
```
df_imdb_crew.head(3)
```

Observation df_df_imdb_crew

This df matches directors and writers to tconst which is primary key in df_imdbbasics and df_imdb_principals.

IMDB principals - df_imfb_principals

In [ ]:
```
df_imdb_principals.info()
```

In [ ]:
```
df_imdb_principals.head(3)
```

Observation df_imdb_principals

This data lists the roles of principals in movies(tconst) to categories and job and characters if they act.
Links to df_imdb_crew, df_imdbbasics, df_imdb_name, df_imdb_ratings.
nconst links to values in the df_imdb_crew and df_imdb_name listing the directors and writers.

IMDB ratings-df_imfb_ratings

In [ ]:
```
df_imdb_ratings.info()
```

In [ ]:
```
df_imdb_ratings.head(3)
```

Observation df_imdb_ratings

Pretty straightforward rating and number of votes. title tconst links to tconst(title) in other imdb tables.

TMB-df_tmb

In [ ]:
```
##The Movie DB
df_tmb.info()
```

In [ ]:
```
df_tmb.tail(3)
```

Observation df_tmb

There are no Nans. This looks to be good data about the movies and genres.
Release date is an object and may be converted to a datatime if to be used. Where to look up genre_ids?

TN The Numbers-df_tn_movie_budget

In [ ]:
```
##The Numbers
df_tn_movie_budget.info()
```

In [ ]:
```
df_tn_movie_budget.head(10)
```

Observation df_tn_movie_budget

There are no Nans. This data is the movies, release date and worldwide gross with production budget. Money data should be converted to int and dollas signs removed Release date is an object and may be converted to a datatime if to be used.

Data Cleaning and Typing

Removing the NanNs, dropping columns which have no importance or too many nans, converting object datatypes to be useful.

Clean df_bom data

In [ ]:
```
df_bom.isna().sum()
```

In [ ]:
```
df_bom.shape
```

In [ ]:
```
df_bom.info()
```

In [ ]:
```
#If NaN setting to 0..
df_bom.fillna(0, inplace=True)
```

In [ ]:
```
df_bom.isna().sum()
```

In [ ]:
```
df_bom.info()
#Need to convert the foreign gross to float
```

A Few values needed to be converted to billions

In [ ]:
```
# When converting to numeric foreign gross at line numbers 1872,1873,1874,2760,
#3079 were in a shorthand billions
# ie 1,131.6 for 1131600000.
# this is the quick fix
df_bom.iloc[[1872],[2]] =1131600000.0
df_bom.iloc[[1873],[2]] =1019400000.0
df_bom.iloc[[1874],[2]] =1163000000.0
df_bom.iloc[[2760],[2]] =1010000000.0
df_bom.iloc[[3079],[2]] =1369500000.0
```

Convert the object number of foreign gross to numeric

In [ ]:
```
#Convert string to numeric values...5 or so of the billions needed to be converted
df_bom['foreign_gross'] = pd.to_numeric(df_bom['foreign_gross'])
```

In [ ]:
```
df_bom.info()
```

In [ ]:
```
#creating the worldwide_gross column from the dommestic and foreign gross
df_bom['worldwide_gross'] = df_bom['domestic_gross'] + df_bom['foreign_gross']
```

In [ ]:
```
df_bom.head()
```

Observation Set Foreign gross to 0 if NaN. If we need we can use the movie budget to look up. Converted money columns to floats. Created worldwide_gross from domestc and foreign.

Clean imdb_name data

In [ ]:
```
df_imdb_name.shape
```

In [ ]:
```
df_imdb_name.isna().sum()
```

In [ ]:
```
#Can drop birth_year and death year
df_imdb_name = df_imdb_name.drop(columns=['birth_year','death_year'])
```

drop if both primary_profession and known_for_titles are both Nan

In [ ]:
```
df_imdb_name.isna().sum()
```

In [ ]:
```
#drop if both primary_profession and known_for_titles are both Nan
col_lst = ['primary_profession', 'known_for_titles']
df_imdb_name.dropna(axis = 0, subset = col_lst, how = 'all', inplace = True)
```

In [ ]:
```
df_imdb_name.isna().sum()
```

Observation lots of NaNs but are supposed to be blank if non applicable. put in holder value(NA, or job or characters? or 0? Dropped 'birth_year','death_year'. If NAN for known_for can drop? Final decision to fill na with 'unknown'

In [ ]:
```
df_imdb_name[df_imdb_name['known_for_titles'].isna()].head()
```

In [ ]:
```
#if there is a Nan in remaining data fill in with unknown
df_imdb_name['known_for_titles'].fillna(value='unknown', inplace=True)
df_imdb_name['primary_profession'].fillna(value='unknown', inplace=True)
```

In [ ]:
```
df_imdb_name.isna().sum()
```

In [ ]:
```
df_imdb_name.shape
```

Clean imdb_akas data

In [ ]:
```
df_imdb_akas.shape
```

In [ ]:
```
df_imdb_akas.info()
```

In [ ]:
```
#Unsure if this data is useful
#df_imdb_akas = df_imdb_akas[df_imdb_akas['is_original_title'] == 1.0]
```

In [ ]:
```
df_imdb_akas.isna().sum()
```

In [ ]:
```
df_imdb_akas.head(3)
```

In [ ]:
```
set(df_imdb_akas['types'])
```

In [ ]:
```
#for NaNs in these columns set to unknown
```

```
col_list = ['region','language','types','attributes']
for col in col_list:

    df_imdb_akas[col].fillna(value='unknown', inplace=True)
```

In [ ]:
```
df_imdb_akas.isna().sum()
```

In [ ]:
```
df_imdb_akas = df_imdb_akas[df_imdb_akas['is_original_title'].notna()]
```

In [ ]:
```
df_imdb_akas.shape
```

Actions In columns 'region','language','types','attributes'set Nan to 'unknown
removed is_original title id Nan

Clean imdb_basics data

In [ ]:
```
df_imdbbasics.shape
```

In [ ]:
```
df_imdbbasics.isna().sum()
```

In [ ]:
```
df_imdbbasics.tail(10)
```

In [ ]:
```
df_imdbbasics.info()
```

In [ ]:
```
#for NaNs in these columns set to unknown
col_list = ['genres','original_title']
for col in col_list:

    df_imdbbasics[col].fillna(value='unknown', inplace=True)
```

In [ ]:
```
#If runtime minutes missing set to 89.5 minutes
df_imdbbasics['runtime_minutes'].fillna(89.5,inplace=True)
```

In [ ]:
```
df_imdbbasics.isna().sum()
```

Actions In columns 'genres','original_title' set Nan to 'unknown
if runtime minutes set to 89.5 if Nan

Clean imdb_crew data

In [ ]:
```
df_imdb_crew.shape
```

In [ ]:
```
df_imdb_crew.isna().sum()
```

In [ ]:
```
df_imdb_crew.head(5)
```

In [ ]:
```
df_imdb_crew.shape
```

In [ ]:
```
df_imdb_crew = df_imdb_crew[df_imdb_crew['directors'].notna()
                            | df_imdb_crew['writers'].notna()]
```

In [ ]:
```
#for NaNs in these columns set to unknown
col_list = ['directors','writers']
for col in col_list:
    df_imdb_crew[col].fillna(value='unknown', inplace=True)
```

In [ ]:
```
df_imdb_crew.info()
```

Actions If directors and writers are NaN drop row(4474 rows). If then if directors is Nan replace with holder 'unknown'. If writers is Nan replace with unknown.

Clean imdb_principals data

In [ ]:
```
df_imdb_principals.shape
```

In [ ]:
```
df_imdb_principals.tail()
```

In [ ]:
```
df_imdb_principals.isna().sum()
```

In [ ]:
```
#for NaNs in these columns set to unknown
col_list = ['job','characters']
for col in col_list:
    df_imdb_principals[col].fillna(value='unknown', inplace=True)
```

In [ ]:
```
df_imdb_principals.info()
```

Actions replaced columns job and characters if Nan to 'unknown'

Clean imdb_ratings data

In [ ]:
```
df_imdb_ratings.shape
```

In [ ]:
```
df_imdb_ratings.isna().sum()
```

In [ ]:
```
df_imdb_ratings.head()
```

In [ ]:
```
df_imdb_ratings.info()
```

Actions All Clean. No missing data in this lookup table

Clean df_tmb data

In [ ]:
```
df_tmb.shape
```

In [ ]:
```
df_tmb.isna().sum()
```

In [ ]:
```
df_tmb.info()
```

In [ ]:
```
df_tmb.head(3)
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Actions: Clean Data Note: Convert release date to datetime when using

Clean df_tn_movie_budget data

In [ ]:
```
df_tn_movie_budget.shape
```

In [ ]:
```
df_tn_movie_budget.isna().sum()
```

In [ ]:
```
df_tn_movie_budget.head(50)
```

In [ ]:
```
for col in df_tn_movie_budget:
    print(col)
```

```
    print(df_tn_movie_budget[col].value_counts(normalize = True)[:5])
    print("==================================")
```

Domectic gross has 9.4 % of zero values. Similiar to worldwide gross.

Will need to address.

look at budget data in bom, rott, tn df_tn_movie_budget[df_tn_movie_budget['domestic_gross'] > 500000000.0]

In [ ]:
```
#Convert the object columns to ints where we can manipulate mathematically
#money string to ints
def money_to_float(df,col):
    df[col] = df[col].astype(str).str.replace("$", "").str.replace(",", "").astype('float')
    return df
```

In [ ]:
```
money_cols = ['production_budget','domestic_gross', 'worldwide_gross']

for col in money_cols:
    df_tn_movie_budget = money_to_float(df_tn_movie_budget,col)
```

In [ ]:
```
df_tn_movie_budget.head(10)
```

In [ ]:
```
df_tn_movie_budget.info()
```

Actions Converted string currrency to float values. Convert release date to datetime when using

Save cleaned files as tidy files

Action: For each of the cleaned DataFrames save the df tidy.csv.

Will not have to keep the original .gz files and can load the cleaned files in EDA.

In [ ]:
```
#dicts of dfname and df
def create_df_dict(namelist,dflist):
    dict_df_names = dict(zip(namelist, dflist))
    return dict_df_names

#takes a dict and saves all to csvs per savepath
def save_dict_tocsv(savepath, dict_dfs, suffix):
    for key,value in dict_dfs.items():
        value.to_csv(path_or_buf = savepath
                     + key +'_tidy' + suffix, encoding='utf8')
```

In [ ]:
```
##List of Cleaned dfs
```

```
dfs_tidy_dict = [df_bom,df_imdb_name,df_imdb_akas,df_imdbbasics,df_imdb_crew,
       df_imdb_principals,df_imdb_ratings,df_tmb,df_tn_movie_budget]
dfs_names = ['df_bom','df_imdb_name','df_imdb_akas','df_imdbbasics','df_imdb_crew',
       'df_imdb_principals','df_imdb_ratings','df_tmb','df_tn_movie_budget']
savepath = '../data/'

dict_cleaned_dfs = create_df_dict(dfs_names, dfs_tidy_dict)
save_dict_tocsv('../data/', dict_cleaned_dfs,'.csv' )
```

EDA in eda_notebook.ipynb

In [ ]:

In [ ]:

Data Exploration

In this workbook we:

1. Reload the cleaned data to DataFrames
2. Perform EDA(Exploraory Data Analysis) to Answer Questions about the business problem
3. Summarize the EDA

Reload Cleaned data

In [6]:
```
#helper to list all csv or type files in a dir
def find_csv_filenames( path_to_dir, suffix=".csv"):
    filenames = listdir(path_to_dir)
    return [ filename for filename in filenames if filename.endswith( suffix ) ]

#creates dataframes for type specified
def create_dfs(path, filelist, suffix=".csv"):
    #Read all the files and store in a dataFrame
    # the data Frames for each file will be listed in a dict
    # where key is the name and value is the df
    dict_csv_files = {}
```

```
    for filename in csvfiles:
        filename_cleaned = filename.replace("_tidy.csv", "").replace(".","_")#cleaning
        filename_df = pd.read_csv(path + filename, index_col = 0, encoding='utf8')
        dict_csv_files[filename_cleaned] = filename_df
    return dict_csv_files

#dicts of dfname and df
def create_df_dict(namelist,dflist):
    dict_df_names = dict(zip(namelist, dflist))
    return dict_df_names

#takes a dict and saves all to csvs per savepath
def save_dict_tocsv(savepath, dict_dfs, suffix):
    for key,value in dict_dfs.items():
        value.to_csv(path_or_buf = savepath
                     + key +'_tidy' + suffix, encoding='utf8')
```

In [7]:
```
#Create csvfiles, tsvfiles and call createdfs dict
path = '../data/'
csvfiles = find_csv_filenames(path)
dict_dfs = create_dfs(path, csvfiles, suffix=".csv" )
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-7-753b01e5d1ec> in <module>
      1 #Create csvfiles, tsvfiles and call createdfs dict
      2 path = '../data/'
----> 3 csvfiles = find_csv_filenames(path)
      4 dict_dfs = create_dfs(path, csvfiles, suffix=".csv" )

<ipython-input-6-44f06c4b8ec7> in find_csv_filenames(path_to_dir, suffix)
      1 #helper to list all csv or type files in a dir
      2 def find_csv_filenames( path_to_dir, suffix=".csv"):
----> 3     filenames = listdir(path_to_dir)
      4     return [ filename for filename in filenames if filename.endswith( suffix ) ]
      5

FileNotFoundError: [WinError 3] The system cannot find the path specified: '../data/'
```

In [ ]:
```
dict_dfs.keys()
```

In [ ]:
```
##Create Working DataFrames
df_bom = dict_dfs['df_bom']
df_imdbbasics = dict_dfs['df_imdbbasics']
df_imdb_akas = dict_dfs['df_imdb_akas']
df_imdb_name = dict_dfs['df_imdb_name']
```

```
df_imdb_crew = dict_dfs['df_imdb_crew']
df_imdb_principals = dict_dfs['df_imdb_principals']
df_imdb_ratings = dict_dfs['df_imdb_ratings']
df_tmb = dict_dfs['df_tmb']
df_tn_movie_budget = dict_dfs['df_tn_movie_budget']
```

Data Exploration

In the EDA (Exploratory Data Analysis) phase, we will work to answer the following question about the business problem by visually answering the data. Our business problem is to deliver actionable insights about the movie industry, specifically types of movies. I approached this Business Problem as defining a movie studio business strategy which leads me to these

Questions:

1. What is success for a feature film? This educates and defines expectations
2. At what level of production budget will we be comfortable investing?
3. What types of feature films genres are we going to make?
4. When should we most optimally release our movies? Are there better months for our releases?
5. How many feature films should we release per year? ie drives initial investment
6. Any correlation to MPA Rating?
7. Who in the industry would be good to work with as producers and directors?
8. Other factors to consider.

What is success for a feature film?

To answer this we should look to analyse data for movies we might consider making.

In industry terms there are four types of production level movies.

1. High Budget: Production budget(PB) is greater than 80 Million US Dollars
2. Medium Budget: PB is between 2 to 80 Million USD
3. Low Bugdet: PB is between 10K and 2 Million USD
4. Micro Budget: PB is under 10K

As a first run in the movie business Microscoft would not want to take a chance on high budget features so we will look at returns in the Medium budget. The average PB(production budget) is right around $65 Million. We will start with that as our cap. Best Return on Investment of Medium budget movies

Lets look at the df_tn_movie_budget data focusing at worldwide gross.

In [ ]:
```
df_tn_movie_budget.info()
```

Lets create a feature for difference between worldwide gross and prod budget
Business terms: Profit = Returned - Investment
Our Data: profit_over_pb = worldwide_gross - production_budget

Let's Calculate percent returned for movies with budgets below $65Mill
Business terms: ROI = Profit / Cost of the investment *100*
Our Data: roi_percent= profit_over_pb / production_budget 100

In [ ]:
```
df_tn_movie_budget['profit_over_pb'] = df_tn_movie_budget['worldwide_gross'] - \
df_tn_movie_budget['production_budget']
```

In [ ]:
```
df_tn_movie_budget['roi_percent'] = round((df_tn_movie_budget['profit_over_pb'] / \
df_tn_movie_budget['production_budget'])*100,1)#round to 1 digit
```

In [ ]:
```
#Sort on roi_percent
df_tn_movie_budget.sort_values(by='roi_percent',ascending = False).head(50)
```

Lets filter to look at movies where budget is less than=$65 mill

In [ ]:
```
df_budget_sub65m = df_tn_movie_budget[df_tn_movie_budget['production_budget'] \
                   <= 65000000.0].sort_values(by='roi_percent', ascending = False)
```

In [ ]:
```
df_budget_sub65m.info()#Note 5012 movies
```

In [ ]:
```
df_budget_sub65m.reset_index(inplace =True)
```

In [ ]:
```
#filer movies greater than 2 Mil
df_budg_2to65mil = df_budget_sub65m[df_budget_sub65m['production_budget'] \
                   >= 2000000.0].sort_values(by='roi_percent', ascending = False)
```

In [ ]:
```
df_budg_2to65mil.info()#note 4231 movies
```

In [ ]:
```
df_budg_2to65mil.reset_index(inplace =True)
```

Lets create a easy human readable feature called **x_times_invest**.

This is equal to our worldwide_gross / PB where as ROI percent is the profit / PB * 100.

In [ ]:
```
df_budg_2to65mil['x_times_invest'] = round(df_budg_2to65mil['worldwide_gross']/df_budg_2to65mil['production_budget'],1)
```

In [ ]:
```
df_budg_2to65mil.head(20)
```

It makes sense to only focus on recent movies. Lets look at the movies from 2010 onward.

In [ ]:
```
df_budg_2to65mil_11yr = df_budg_2to65mil[pd.to_datetime(df_budg_2to65mil['release_date']).dt.year >= 2010]
```

In [ ]:
```
df_budg_2to65mil_11yr.reset_index(inplace =True)
```

In [ ]:
```
df_budg_2to65mil_11yr.describe()
```

What is our target?

Based on industry research, movies dont truly turn a profit until the 2.0 to 2.5 times PB mark due to marketing and distributors.

Note: The **median** value of medium budget movies (2-65mil) is **61.5% ROI or 1.6 x the investment.**

Lets set our target and define success as movies with 150% ROI_percent or wwgross 2.5 times the investment.

Example:PB is 5000000, **2.5** times is 12,500,000 for worldwide_gross

Profit would be 7500000 ROI%=7500000 /5000000 *100 =**150%**

Note: There are many outlers in this data. These movies are extremly successful. Graph only to 1750%. Max was 6000% ROI%

In [ ]:
```
#Insert graph of all movies roi with red line at 2.5xPB or 150% roi

sns.set(style="whitegrid")
fig, ax1 = plt.subplots(figsize=(13,3),)

boxplot = sns.boxplot(ax=ax1, x=df_budg_2to65mil_11yr["roi_percent"],color='skyblue')
#sns.stripplot(ax=ax1, x=df_budg_2to65mil_11yr["roi_percent"],color='skyblue')
ax1.set(xlabel = 'ROI Percent', title='Boxplot of ROI Percent with Success Line')
ax1.axvline(150, ls='--',color='red',label='Success Line 150%')
ax1.set_xlim(xmin=-250,xmax=1750)
ax1.legend(loc='upper right')
```

```
ax1.text(x=-10,y=0,s="65% Fail")
plt.show();
```

Note: How many movies with roi_percent at >=150% ? Only 533 from 1485. Only **35% were successful. 65% fail**.

In [ ]:
```
#Lets only analyze the movies with >=150% ROI(533 movies)
df_budg_success_11yrs = df_budg_2to65mil_11yr[df_budg_2to65mil_11yr['roi_percent'] >= 150]
```

533 successful movies INSERT graph of top 10

In [ ]:
```
df_budg_success_11yrs.head(10)
```

In [ ]:
```
df_budg_success_11yrs = df_budg_success_11yrs.drop(columns=['level_0','index'])
```

In [ ]:
```
df_budg_success_11yrs.describe()
```

In [ ]:
```
#save successful movies to file
df_budg_success_11yrs.to_csv(path_or_buf = path + 'budg_success_11yrs')
```

Conclusion: The Percent RoI for movies in the medium budget range from 2 - 65 M USD has a mean of 586% return but median is 61.5%. The mean is skewed due to the number of outliers. Most movies do not succeed. In this data, **65%** do not succeed. Considering marketing and distribution, A successful movie should return at least 2.5 times the production budget, ie 150% Roi. Going forward we will only look at those movies.

Question: What level of production budget will we be comfortable investing?

Of the 533 Successful Films in medium budget range since 2010, lets look into the production budgets of those.

In [8]:
```
df_budg_success_11yrs.describe()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-1dbfa29fe814> in <module>
----> 1 df_budg_success_11yrs.describe()

NameError: name 'df_budg_success_11yrs' is not defined
```

In [ ]:
```
#Histogram of Production budget bins

sns.set(style="whitegrid")
fig, ax1 = plt.subplots(figsize=(18,6),sharex=True ,sharey=True)

histplot = sns.histplot(ax=ax1, x=df_budg_success_11yrs["production_budget"],
```

```
                        color='skyblue',bins=7,label='Bin Range of $9M')

ax1.set_xlabel('Production Budgets', fontsize=15)
ax1.set_ylabel('Number of Films', fontsize=15)
ax1.set_title('533 Successful Films: Histogram of Production Budgets',fontsize=20)
#Set the Average Line
ax1.axvline(df_budg_success_11yrs["production_budget"].mean(), ls='--',
            color='green',label='$22.6M Mean')
ax1.set(xlim = (0,67000000))
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax1.get_xticks().tolist()
ax1.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax1.set_xticklabels(['{:,.2f}'.format(x/1000000) + 'M' for x in ticks_loc])

ax1.set(xlim = (0,67000000))
ax1.legend(loc='upper right')

fig.savefig('../images/ProdBudg.png', bbox_inches='tight')


plt.show();
```

Conclusion: Most of the movies are in the 2M to 11M and 11M to 20M bin ranges with a steady decline as budgets increase. We do not need to spend a high amount on budget to make a successful film. We need to look at *genres* of the movies to determine more. Looking at the Average ROI per range would also be informative.

Question: What types of feature film genres should we make?

Lets pull in genre information for the movies. This has been noted in the df_imdbbasics. We will join this with out budget data.

In [ ]:
```
df_imdbbasics.head(3)
```

In [ ]:
```
df_budg_success_11yrs.info()
```

In [ ]:
```
#We will do a left join on the successful movies and the imdb basics
#keying on movie name and year of release
df_budget_genres = pd.merge(df_budg_success_11yrs, df_imdbbasics,
                            left_on=  ['movie',
                                       pd.to_datetime(df_budg_success_11yrs['release_date']).dt.year],
```

```
                                right_on= ['primary_title','start_year'],
                                how = 'left')
```

In [ ]:
```
df_budget_genres.info()
```

In [ ]:
```
#We have to exclude where the genres were unknown or Nan
df_budget_genres = df_budget_genres[df_budget_genres.genres != 'unknown']
```

In [ ]:
```
df_budget_genres = df_budget_genres[df_budget_genres.primary_title.notna()]
```

In [ ]:
```
#We have matching genre info for 428 successful movies
df_budget_genres.info()
```

In [ ]:
```
#Onlyrun this code once or the genre lis will be in a list itself
#genres is a list of upto 3 genres, splitting it to get the individual genres
df_budget_genres['genres'] = df_budget_genres['genres'].astype(str).apply(lambda x: x.split(",") if x else x)
df_budget_genres.tail(10)
```

In [ ]:
```
df_budget_genres.info()
```

In [ ]:
```
#a set of distinct genres in the df
all_genres = set()
for genres in df_budget_genres['genres']:
    if genres:
        all_genres.update(genres)
```

In [ ]:
```
#Listing of all distinct genres
all_genres
```

In [ ]:
```
#adding cols with zeros for all the genres we have. Will modify genre to 1
#if the film is of that genre.
for genre in all_genres:
    df_budget_genres[genre] = np.zeros(shape=df_budget_genres.shape[0])

df_budget_genres.head()
```

In [ ]:
```
#setting the genre to be 1 if the film is of that genre
for index, row in df_budget_genres.iterrows():
    if row['genres']:
        for genre in row['genres']:
            df_budget_genres.loc[index, genre] = 1
```

```
df_budget_genres.head()
```

In [ ]:
```
len(all_genres)
```

In [ ]:
```
#checking the counts for all different genres
#all_genres
for col in all_genres:
    print(f'Viewing values in col: {col}')
    print(f'Top 5 values:\n{df_budget_genres[col].value_counts()}')
```

In [ ]:
```
#making a list of all genres
cols = list(df_budget_genres.columns)
genre_cols = cols[14:]

#getting a dict with genre counts
genre_count = {}
for col in genre_cols:
    count = np.sum(df_budget_genres[col] == 1).sum()
    genre_count[col] = count
```

In [ ]:
```
genre_count
```

In [ ]:
```
keys = list(genre_count.keys())
values = list(genre_count.values())
```

In [ ]:
```
#Histogram of Genres Counts

sns.set(style="darkgrid")
fig, ax1 = plt.subplots(figsize=(12,4),sharex=True ,sharey=True)

histplot = sns.barplot(y = keys, x = values, color = 'skyblue',
                       label='Count of Genre Tags')

ax1.set_xlabel('Genre Count', fontsize=15)
ax1.set_ylabel('Film Genres', fontsize=15)
ax1.set_title('Count of All Distinct Genres,All Ratings',fontsize=20)
#Set the Average Line

ax1.set(xlim = (0,250))

ax1.legend(loc='upper right')
plt.show();
```

The 6 most popular genres are Drama, Comedy, Thriller, Horror, Action and Romance.

Grouping by genres to look at sum, mean and max related to worldwide gross and roi_percent.

In [ ]:
```
#Total Gross by Genre
ww_gross = {}
for genre in all_genres:
    grouped = df_budget_genres.groupby(by =''.join(genre)).sum()
    ww_gross[genre] = grouped.iloc[1]['worldwide_gross']
```

In [ ]:
```
#Average Gross by Genre
ww_gross_avg = {}
for genre in all_genres:
    grouped = df_budget_genres.groupby(by =''.join(genre)).mean()
    ww_gross_avg[genre] = grouped.iloc[1]['worldwide_gross']
```

In [ ]:
```
#Max Roi% by Genre
max_roi_genre = {}
for genre in all_genres:
    grouped = df_budget_genres.groupby(by =''.join(genre)).max()
    max_roi_genre[genre] = grouped.iloc[1]['roi_percent']
```

In [ ]:
```
#Average Roi% by Genre
avg_roi_genre = {}
for genre in all_genres:
    grouped = df_budget_genres.groupby(by =''.join(genre)).mean()
    avg_roi_genre[genre] = grouped.iloc[1]['roi_percent']
```

In [ ]:
```
#Average Prod Budget by Genre
avg_prod_budget = {}
for genre in all_genres:
    grouped = df_budget_genres.groupby(by =''.join(genre)).mean()
    avg_prod_budget[genre] = grouped.iloc[1]['production_budget']
```

In [ ]:
```
#Min Prod Budget by Genre
min_prod_budget = {}
for genre in all_genres:
    grouped = df_budget_genres.groupby(by =''.join(genre)).min()
    min_prod_budget[genre] = grouped.iloc[1]['production_budget']
```

In [ ]:
```
ww_gross_avg
```

In [ ]:
```
ww_gross
```

In [9]:
```
max_roi_genre
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-78d015860607> in <module>
----> 1 max_roi_genre

NameError: name 'max_roi_genre' is not defined
```

In [ ]:
```
avg_roi_genre
```

In [ ]:
```
#Genre Grid
#This graph of subplots could be refactored to a function and could be reused
sns.set(style="darkgrid")
fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6)) = plt.subplots(figsize=(10,14),
                                                         nrows=3,ncols=2, )

df_items1 = pd.DataFrame(ww_gross.items())
barplot = sns.barplot(data=df_items1, x=1, y=0,ax=ax1, color = 'skyblue',)
ax1.set(xlabel = 'Worldwide Gross in Billions ',
        ylabel='Genre', title='Total Gross per Genre');
ax1.set_ylabel('Film Genres', fontsize=15)
ax1.set_title('Total Gross per Genre',fontsize=15)
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax1.get_xticks().tolist()
ax1.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax1.set_xticklabels(['{:,.1f}'.format(x/1000000000) + 'B' for x in ticks_loc])


df_items2 = pd.DataFrame(ww_gross_avg.items())
ax2 = sns.barplot(data=df_items2, x=1, y=0,ax=ax2,color = 'skyblue')
ax2.set(xlabel = 'Worldwide Gross in Millions ', ylabel='Genre',
        title='Average Gross per Genre');

ax2.set_ylabel('Film Genres', fontsize=15)
ax2.set_title('Average Gross per Genre',fontsize=15)
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax2.get_xticks().tolist()
ax2.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax2.set_xticklabels(['{:,.0f}'.format(x/1000000) + 'M' for x in ticks_loc])


df_items3 = pd.DataFrame(max_roi_genre.items())
ax3 = sns.barplot(data=df_items3, x=1, y=0, ax=ax3,color = 'skyblue')
```

```
ax3.set(xlabel = '% ROI', ylabel='Film Genre', title='Max Roi Percent per Genre');
ax3.set_ylabel('Film Genres', fontsize=15)
ax3.set_title('Max Roi Percent per Genre',fontsize=15)

df_items4 = pd.DataFrame(avg_roi_genre.items())
ax4 = sns.barplot(data=df_items4, x=1, y=0, ax=ax4, color = 'skyblue')
ax4.set(xlabel = '% ROI', ylabel='Film Genre', title='Avg Roi Percent per Genre');
ax4.set_ylabel('Film Genres', fontsize=15)
ax4.set_title('Avg Roi Percent per Genre',fontsize=15)
# Save just the portion _inside_ the second axis's boundaries
extent = ax4.get_window_extent().transformed(fig.dpi_scale_trans.inverted())

# Pad the saved area by 10% in the x-direction and 20% in the y-direction
fig.savefig('../images/ax4_genre.png', bbox_inches=extent.expanded(1.1, 1.2))


df_items5 = pd.DataFrame(min_prod_budget.items())
ax5 = sns.barplot(data=df_items5, x=1, y=0, ax=ax5, color = 'skyblue')
ax5.set(xlabel = 'Production Budget in Millions', ylabel='Genre', title='Min Production per Genre')
ax5.set_ylabel('Film Genres', fontsize=15)
ax5.set_title('Min Production per Genre',fontsize=15)
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax5.get_xticks().tolist()
ax5.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax5.set_xticklabels(['{:,.2f}'.format(x/1000000) + 'M' for x in ticks_loc])


df_items6 = pd.DataFrame(avg_prod_budget.items())
ax6 = sns.barplot(data=df_items6, x=1, y=0, ax=ax6, color = 'skyblue')
ax6.set(xlabel = 'Production Budget in Millions', ylabel='Genre', title='Avg Production per Genre')
ax6.set_ylabel('Film Genres', fontsize=15)
ax6.set_title('Avg Production per Genre',fontsize=15)
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax6.get_xticks().tolist()
ax6.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax6.set_xticklabels(['{:,.2f}'.format(x/1000000) + 'M' for x in ticks_loc])

plt.subplots_adjust(wspace=0.8,hspace=.4)
plt.show();
```

In [ ]:
```
#Observations---Horror Mystery Thriller are top3 Average Return on budget.  Thriller is one of the top 3 in count
```

In [ ]:
```

```

In [ ]:
```
#This graph of subplots could be refactored to a function and could be reused
sns.set(style="darkgrid")
fig, ax4 = plt.subplots(figsize=(6,6),
                                                nrows=1,ncols=1, )
df_items4 = pd.DataFrame(avg_roi_genre.items())
ax4 = sns.barplot(data=df_items4, x=1, y=0, ax=ax4, color = 'skyblue')
ax4.set(xlabel = '% ROI', ylabel='Film Genre', title='Avg Roi Percent per Genre');
ax4.set_ylabel('Film Genres', fontsize=15)
ax4.set_title('Avg Roi Percent per Genre',fontsize=15)
# Save just the portion _inside_ the second axis's boundaries
extent = ax4.get_window_extent().transformed(fig.dpi_scale_trans.inverted())

# Pad the saved area by 10% in the x-direction and 20% in the y-direction
fig.savefig('../images/ax4_genre.png', bbox_inches=extent.expanded(1.1, 1.2))
```

In [ ]:
```

```

In [ ]:
```

```

Conclusion: Higher returns are with the Horror, Mystery, and Thriller genres.

How does MPAA Rating correlate?

In [ ]:
```
#Lets Add in Movie MPAA Rating and exclude R to protect our parent name brand
#MPAA Rating is in the
df_budget_genres
```

In [ ]:
```
df_mpaa_ratings = pd.read_csv('../data/tn_mpaa_ratings.csv',index_col = 0, encoding='utf8')
```

In [ ]:
```
df_mpaa_ratings.head(10)
```

In [ ]:
```
#We will do a left join on the successful movies and the imdb basics
#keying on movie name and year of release
df_budget_genre_ratings = pd.merge(df_budget_genres, df_mpaa_ratings,
                            left_on=  ['movie',
                                       pd.to_datetime(df_budget_genres['release_date']).dt.year],
                            right_on= ['Title',
                                       pd.to_datetime(df_mpaa_ratings['Released']).dt.year],
                            how = 'left')
```

In [ ]:
```
df_budget_genre_ratings = df_budget_genre_ratings[df_budget_genre_ratings['mpaa_rating'].notna()]
```

```
df_budget_genre_ratings.head()
```

In [ ]:

In [ ]:
```
df_budget_genre_ratings.drop(columns=['id', 'key_1','primary_title',
                                      'original_title','Released',
                                      'ProductionBudget',
                                      'WorldwideBox Office','runtime_minutes'],
                             inplace=True)
```

In [ ]:
```
df_budget_genre_ratings.reset_index(inplace=True)
```

In [ ]:
```
df_budget_genre_ratings
```

In [ ]:
```
df_budget_genre_ratings.drop(columns=['index'], inplace=True)
```

In [ ]:
```
df_budget_genre_ratings.head()
```

In [ ]:
```
df_budget_genre_ratings.mpaa_rating.value_counts()
```

In [ ]:
```
ratings = ['R','PG-13','PG','G']
```

In [ ]:
```
ratingsvalues = list(df_budget_genre_ratings.mpaa_rating.value_counts())
```

In [ ]:
```
ratingsvalues
```

In [ ]:
```
#Histogram of Production budget bins

sns.set(style="darkgrid")
fig, ax1 = plt.subplots(figsize=(8,3))

histplot = sns.barplot(y = ratings, x = ratingsvalues, color = 'skyblue',
                       label='Count of MPAA Rating')

ax1.set_xlabel('MPAA Ratings Count', fontsize=15)
ax1.set_ylabel('MPAA Ratings', fontsize=15)
ax1.set_title('Count of All MPAA Ratings',fontsize=20)
#Set the Average Line

#ax1.set(xlim = (0,250))

#ax1.legend(loc='upper right')
plt.show();
```

In [ ]:
```
df_budget_genre_ratings
```

In [ ]:
```
    grouped = df_budget_genre_ratings.groupby(by ='mpaa_rating').mean()
```

In [ ]:
```
    avg_roi_by_rating[ratings] = grouped.iloc[1]['roi_percent']
```

In [ ]:
```
avg_roi_by_rating
```

In [ ]:
```
df_not_R = df_budget_genre_ratings[df_budget_genre_ratings['mpaa_rating'] != 'R']
```

In [ ]:
```
df_not_R
```

In [ ]:
```
#making a list of all columns
colsnotr = list(df_not_R.columns)
```

In [ ]:
```
colsnotr
```

In [10]:
```
genre_colsnotr = cols[14:36]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-da766c23150f> in <module>
----> 1 genre_colsnotr = cols[14:36]

NameError: name 'cols' is not defined
```

In [ ]:
```
df_not_R
```

In [ ]:
```
genre_colsnotr = cols[14:36]

#getting a dict with genre counts
genre_countnotr = {}
for col in genre_cols:
    count = np.sum(df_not_R[col] == 1).sum()
    genre_countnotr[col] = count
```

In [ ]:
```
genre_countnotr
```

In [ ]:
```
keys_notr = list(genre_countnotr.keys())
values_notr = list(genre_countnotr.values())
```

In [ ]:
```
#Histogram of Genres Counts

sns.set(style="darkgrid")
fig, ax1 = plt.subplots(figsize=(12,4),sharex=True ,sharey=True)

histplot = sns.barplot(y = keys_notr, x = values_notr, color = 'skyblue',
                       label='Count of Genre Tags')

ax1.set_xlabel('Genre Count', fontsize=15)
ax1.set_ylabel('Film Genres', fontsize=15)
ax1.set_title('Count of All Distinct Genres not R rated',fontsize=20)
#Set the Average Line

ax1.set(xlim = (0,140))

ax1.legend(loc='upper right')
plt.show();
```

In [ ]:
```

```

In [ ]:
```
#Average Roi% by Genre
max_roi_genrenotr= {}
for genre in all_genresnotr:
    grouped = df_not_R.groupby(by =''.join(genre)).max()
    max_roi_genrenotr[genre] = grouped.iloc[0]['roi_percent']
```

In [ ]:
```
#Average Roi% by Genre
avg_roi_genrenotr= {}
for genre in all_genresnotr:
    grouped = df_not_R.groupby(by =''.join(genre)).mean()
    avg_roi_genrenotr[genre] = grouped.iloc[0]['roi_percent']
```

In [ ]:
```
#Average Gross by Genre
ww_gross_avgnotr = {}
for genre in all_genresnotr:
    grouped = df_not_R.groupby(by =''.join(genre)).mean()
    ww_gross_avg[genre] = grouped.iloc[0]['worldwide_gross']
```

In [ ]:
```
#Average PB by Genre
pb_avgnotr = {}
for genre in all_genresnotr:
    grouped = df_not_R.groupby(by =''.join(genre)).mean()
    pb_avgnotr[genre] = grouped.iloc[0]['production_budget']
```

In [ ]:
```
max_pb_genrenotr = {}
for genre in all_genresnotr:
    grouped = df_not_R.groupby(by =''.join(genre)).max()
    max_pb_genrenotr[genre] = grouped.iloc[0]['production_budget']
```

In [ ]:
```
max_roi_genrenotr
```

In [ ]:
```
#a set of distinct genres in the df
all_genresnotr = set()
for genres in df_not_R['genres']:
    if genres:
        all_genresnotr.update(genres)
```

In [ ]:
```
#Genre Grid
#This graph of subplots could be refactored to a function and could be reused
sns.set(style="darkgrid")
fig, ((ax3, ax4), (ax5, ax6)) = plt.subplots(figsize=(10,12),
                                             nrows=2,ncols=2, )
max_roi_genrenotr
df_items3 = pd.DataFrame(max_roi_genrenotr.items())
ax3 = sns.barplot(data=df_items3, x=1, y=0, ax=ax3, color = 'skyblue')
ax3.set(xlabel = '% ROI', ylabel='Film Genre', title='Max Roi Percent per Genre');
ax3.set_ylabel('Film Genres', fontsize=15)
ax3.set_title('Max Roi Percent per Genre',fontsize=15)



df_items4 = pd.DataFrame(avg_roi_genrenotr.items())
ax4 = sns.barplot(data=df_items4, x=1, y=0, ax=ax4, color = 'skyblue')
ax4.set(xlabel = '% ROI', ylabel='Film Genre', title='Avg Roi Percent per Genre');
ax4.set_ylabel('Film Genres', fontsize=15)
ax4.set_title('Avg Roi Percent per Genre',fontsize=15)

df_items5 = pd.DataFrame(max_pb_genrenotr.items())
ax5 = sns.barplot(data=df_items5, x=1, y=0, ax=ax5, color = 'skyblue')
ax5.set(xlabel = 'PB', ylabel='Film Genre', title='Max PB per Genre');
ax5.set_ylabel('Film Genres', fontsize=15)
ax5.set_title('Max PB per Genre',fontsize=15)
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax5.get_xticks().tolist()
ax5.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax5.set_xticklabels(['{:,.0f}'.format(x/1000000) + 'M' for x in ticks_loc])

df_items6 = pd.DataFrame(pb_avgnotr.items())
ax6 = sns.barplot(data=df_items6, x=1, y=0, ax=ax6, color = 'skyblue')
```

```
ax6.set(xlabel = 'Production Budget in Millions', ylabel='Genre', title='Avg Production per Genre')
ax6.set_ylabel('Film Genres', fontsize=15)
ax6.set_title('Avg Production per Genre',fontsize=15)
# fixing xticks warning with matplotlib.ticker "FixedLocator"
ticks_loc = ax6.get_xticks().tolist()
ax6.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
ax6.set_xticklabels(['{:,.0f}'.format(x/1000000) + 'M' for x in ticks_loc])

plt.subplots_adjust(wspace=0.8,hspace=.4)
plt.show();
```

In [ ]:
```
sns.set(style="darkgrid")
fig, ax4 = plt.subplots(figsize=(6,6),nrows=1,ncols=1, )

df_items4 = pd.DataFrame(avg_roi_genrenotr.items())
ax4 = sns.barplot(data=df_items4, x=1, y=0, ax=ax4, color = 'skyblue')
ax4.set(xlabel = '% ROI', ylabel='Film Genre', title='Avg Roi Percent per Genre');
ax4.set_ylabel('Film Genres', fontsize=15)
ax4.set_title('Avg Roi Percent per Genre',fontsize=15)
```

Conclusion: Genres higher average returns are with R rating are Horror, Mystery and Thrillers. Excluding R ratings, Comedy has a higher ROI% closely followed by Adventure, Crime, Action, Sport, Romance. Anything really except Horror, Drama are worst when not R rated but not by much.

Question: When should we most optimally release our movies? Are there better months for our releases?

In answering the question When to Release the movie lets look to see when previous movies were released. Lets make a column for release month.

In [ ]:
```
df_budget_genres.head()
```

In [ ]:
```
df_budget_genres['release_month'] = pd.to_datetime(df_budget_genres.release_date).dt.strftime('%b')
#df_budget_genres['release_month'] = pd.to_datetime(df_budget_genres.release_date).dt.month
```

In [ ]:
```
df_budget_genres.head()
```

In [ ]:
```
df_budget_genres
```

In [ ]:
```
pd.to_datetime(df_budget_genres['release_date']).dt.year
```

In [ ]:
```
df_budget_genres['month'] = pd.to_datetime(df_budget_genres['release_date']).dt.month
```

In [ ]:
```
df_budget_genres.sort_values(by='month', inplace=True)
```

In [ ]:
```
##Histplot of movies release in month
sns.set(style="darkgrid")
fig, (ax1, ax2) = plt.subplots(figsize=(16,6),nrows =2, ncols=1,sharex=True ,sharey=False)
ax1 = sns.histplot(df_budget_genres, x="release_month",bins=12,shrink=.8, ax=ax1)
ax1.set(xlabel = 'Month of Film Release', ylabel='Number of Films', title='Count per Month')

sns.boxplot(x='release_month',y='roi_percent',data=df_budget_genres,ax=ax2,
            palette='cool',fliersize=0)
ax2.set(xlabel = 'Month vs ROI %', ylabel='ROI %', )
ax2.set_ylim(top=1600)
ax2.axhline(y=570)

plt.subplots_adjust(wspace=0.8, hspace=0.05)
plt.show();
```

Conclusion: Some very good high returns on investment occurred in the months of October and December. The median returns per month are similar and all under the 10 times investment. There is a down trend with September and May not having huge gains. October could be closely related with Halloween and Horror and/or Thriler movies. Safe months are June, July, Aug, Oct Nov.

Question: How many feature films should we release per year? ie drives initial investment

How many movies a year? Lets Look at the successful studios number a year.

df_bom has studio info...We can join on budget genres to get month and year.

In [ ]:
```
df_bom.head()
```

In [ ]:
```
df_budget_genres.info()
```

In [ ]:
```
df_studios_bud_genres = pd.merge(df_budget_genres, df_bom,
                            left_on=  ['movie',pd.to_datetime(df_budget_genres['release_date']).dt.year],
```

```
                                right_on= ['title','year'],
                                how = 'left')
```

In [ ]:
```
df_studios_bud_genres[df_studios_bud_genres.studio.isna()]
```

In [ ]:
```
df_bom[df_bom.index== 'Get Out']
```

In [ ]:
```
df_studios_bud_genres = df_studios_bud_genres[df_studios_bud_genres.studio.notna()]
```

In [ ]:
```
#a set of distinct genres in the df
all_studios = set(df_studios_bud_genres['studio'])
```

In [ ]:
```
all_studios
```

In [ ]:
```
#getting a dict with Studio counts
studio_count = {}
for col in genre_cols:
    count = np.sum(df_budget_genres[col] == 1).sum()
    genre_count[col] = count
```

In [11]:
```
df_studios_bud_genres['studio'].value_counts()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-d47e79e94f38> in <module>
----> 1 df_studios_bud_genres['studio'].value_counts()

NameError: name 'df_studios_bud_genres' is not defined
```

In [ ]:
```
studio_count = dict(df_studios_bud_genres['studio'].value_counts())
```

In [ ]:
```
studio_count
```

In [ ]:
```
df_bom_studio = df_bom.groupby(by=['studio','year']).count()
```

In [ ]:
```
df_bom_studio
```

In [ ]:
```
df_bom_studio_pivott = df_bom.pivot_table(index='studio' ,values='worldwide_gross',
                                          columns='year',
                                          margins=True,margins_name='count',
                                          aggfunc='count',fill_value=0)
```

In [ ]:
```
df_bom_studio_pivott.sort_values(['count'], ascending=[False], inplace=True)
df_bom_studio_pivott
```

In [ ]:
```
type(df_bom_studio)
```

In [ ]:
```
testpiv = df_bom_studio_pivott[1:21]
```

In [ ]:
```
mask=[False, False, False, False, False, False,
                        False, False, False,True]
# use Seaborn styles
sns.set()
fig, ax9 = plt.subplots(figsize=(12, 8))

ax9 = sns.heatmap(annot=True, fmt="d", linewidths=.5,data=testpiv, ax=ax9, cmap='coolwarm',mask=mask)
ax9.set(xlabel = 'Film Year', ylabel='Film Studio', title='Studio Films per Year ')
plt.show()
```

In [ ]:
```
# of the Successful movies number per year
#df_studios_bud_genres['studio'].value_counts())
df_studio_budget_pivott = df_studios_bud_genres.pivot_table(index='studio' ,values='worldwide_gross_x',
                                        columns='year',
                                        margins=True,margins_name='count',
                                        aggfunc='count',fill_value=0)
```

In [ ]:
```
df_studio_budget_pivott.sort_values(['count'], ascending=[False], inplace=True)
```

In [ ]:
```

```

In [ ]:
```
mask=[False, False, False, False, False, False,
                        False, False, False,True]
# use Seaborn styles
sns.set()
fig, ax10 = plt.subplots(figsize=(12, 8))

ax10 = sns.heatmap(annot=True, fmt="d", linewidths=.5,data=df_studio_budget_pivott[1:21], ax=ax10, cmap='coolwarm',mask=m
ax10.set(xlabel = 'Film Year', ylabel='Film Studio', title='Successful Films per Year ')
fig.savefig('../images/SuccessFilmsYear.png', bbox_inches='tight')
plt.show()
```

In [ ]:
```

```

Conclusion: With a max of 9 successful films a year is spectacular, a safe bet looks to be 3 to 5 starting out.

Question: What are the sources of the movies?

We scraped some data from TheNumbers which had the source material.

In [ ]:
```
df_movie_source = pd.read_csv('../data/tn_moviesource.csv',index_col = 0, encoding='utf8')
```

In [ ]:
```
df_movie_source['Source'].value_counts()
```

In [ ]:
```
df_budget_genre_ratings_source = pd.merge(df_budget_genre_ratings, df_movie_source,
                            left_on=  ['movie'],
                            right_on= ['Title'],
                            how = 'left')
```

In [ ]:
```
df_budget_genre_ratings_source.columns
```

In [ ]:
```
df_budget_genre_ratings_source.drop(columns=['Title','Released_y','Released_x', 'Title_y',
                                             'Source_x','Released_x', 'Title_y', 'Source_x'],inplace=True)
```

In [ ]:
```
df_budget_genre_ratings_source
```

In [ ]:
```
sourcecounts = df_budget_genre_ratings_source['Source_y'].value_counts(ascending=False).to_frame()
```

In [ ]:
```
sourcecounts
```

Conclusion: Original Screenplay 219 , Based on book or short story 70, Real Life Events 48

Conclusions

Empower Studios Portfolio Strategy includes: Either

Embrace R

Horror Mystery Thriller Highest ROI%

Have the highest average return on Investment.
or
Produce No R Drama, Comedy and Romance, or any except Horror aka Disney Approach

Both Plans include
Produce 5 to 8 films per year in the <$20M budget Range
Release in Summer or late Fall
Looking for 50% Original Content 50% Book Source or Factual Events other

Next Steps

Analysis of Successful Producers, Directors, Cinematographers, Actors
Associating critical rating with success
Academy Awards nominations with successful box office