Capstone Project I - In-Depth Analysis

October 29, 2019

1 Introduction

Movie companies lose great amounts of money on unsuccessful movies every year. Having past data about unsuccessful movies and utilizing machine learning algorithms can greatly help those companies make better business decisions. This project will utilize a movies' dataset to explore any interesting trends and try to build successful models to predict movie revenues accurately.

Here, we will try to draw insights from the dataset by visualizing the relations between different features of the movies' dataset and revenue of the movie which represents how successful is the movie. These visualizations will be done over many steps due to the different types of feature columns and the type of data being explored. For some data columns, we will use boxplots and swarm plots to reveal more information about the distribution of data. These exploratory steps will help guide next analytical and machine learning steps later in the project.

2 Libraries Import

Here, we will import essential libraries for data wrangling. Others will be imported later as needed

[10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

3 Dataset Loading Into Environment

[95]:
```
df = pd.read_csv('movies.csv', encoding = "ISO-8859-1")
# Explore the 1st 5 rows of dataset
df.head()
```

[95]:
```
       budget                              company country         director  \
0   8000000.0          Columbia Pictures Corporation     USA       Rob Reiner
1   6000000.0                     Paramount Pictures     USA      John Hughes
2  15000000.0                     Paramount Pictures     USA       Tony Scott
3  18500000.0  Twentieth Century Fox Film Corporation     USA    James Cameron
4   9000000.0                  Walt Disney Pictures     USA   Randal Kleiser
```

```
       genre        gross                      name rating    released  \
0  Adventure   52287414.0               Stand by Me      R  1986-08-22
1     Comedy   70136369.0  Ferris Bueller's Day Off  PG-13  1986-06-11
2     Action  179800601.0                   Top Gun     PG  1986-05-16
3     Action   85160248.0                    Aliens      R  1986-07-18
4  Adventure   18564613.0   Flight of the Navigator     PG  1986-08-01

   runtime  score               star   votes          writer  year
0       89    8.1        Wil Wheaton  299174    Stephen King  1986
1      103    7.8  Matthew Broderick  264740     John Hughes  1986
2      110    6.9         Tom Cruise  236909        Jim Cash  1986
3      137    8.4   Sigourney Weaver  540152  James Cameron  1986
4       90    6.9        Joey Cramer   36636  Mark H. Baker  1986
```

[96]:
```
df.describe(include='all')
```

[96]:
```
              budget             company country      director   genre  \
count   6.820000e+03                6820    6820          6820    6820
unique           NaN                2179      57          2759      17
top              NaN  Universal Pictures     USA  Woody Allen  Comedy
freq             NaN                 302    4872            33    2080
mean    2.458113e+07                 NaN     NaN           NaN     NaN
std     3.702254e+07                 NaN     NaN           NaN     NaN
min     0.000000e+00                 NaN     NaN           NaN     NaN
25%     0.000000e+00                 NaN     NaN           NaN     NaN
50%     1.100000e+07                 NaN     NaN           NaN     NaN
75%     3.200000e+07                 NaN     NaN           NaN     NaN
max     3.000000e+08                 NaN     NaN           NaN     NaN

               gross   name rating    released     runtime        score  \
count   6.820000e+03   6820   6820        6820  6820.00000  6820.000000
unique           NaN   6731     13        2403         NaN          NaN
top              NaN  Pulse      R  1991-10-04         NaN          NaN
freq             NaN      3   3392          10         NaN          NaN
mean    3.349783e+07    NaN    NaN         NaN   106.55132     6.374897
std     5.819760e+07    NaN    NaN         NaN    18.02818     1.003142
min     7.000000e+01    NaN    NaN         NaN    50.00000     1.500000
25%     1.515839e+06    NaN    NaN         NaN    95.00000     5.800000
50%     1.213568e+07    NaN    NaN         NaN   102.00000     6.400000
75%     4.006534e+07    NaN    NaN         NaN   115.00000     7.100000
max     9.366622e+08    NaN    NaN         NaN   366.00000     9.300000

                star         votes       writer         year
count           6820  6.820000e+03         6820  6820.000000
unique          2504           NaN         4199          NaN
top     Nicolas Cage           NaN  Woody Allen          NaN
freq              42           NaN           32          NaN
mean             NaN  7.121952e+04          NaN  2001.000293
```

```
std              NaN  1.305176e+05          NaN     8.944501
min              NaN  2.700000e+01          NaN  1986.000000
25%              NaN  7.665250e+03          NaN  1993.000000
50%              NaN  2.589250e+04          NaN  2001.000000
75%              NaN  7.581225e+04          NaN  2009.000000
max              NaN  1.861666e+06          NaN  2016.000000
```

4 Data Cleaning

[97]:
```python
# Check the number of Nans
df.isnull().sum()
```

[97]:
```
budget      0
company     0
country     0
director    0
genre       0
gross       0
name        0
rating      0
released    0
runtime     0
score       0
star        0
votes       0
writer      0
year        0
dtype: int64
```

[98]:
```python
# There are zero Nans or nulls
# According to the dataset description, there are some movies with unknown␣
 ↪budgets given as 0, let's count how many of them
df.budget[df.budget==0].count()
```

[98]:
```
2182
```

There are about 2200 movies with missing budget. Because budget may be a critical variable in estimating gross, we will filter out those movies for now instead of substituting their budget with a statistic. Later, we will get back to those movies, estimate their budgets by other means, and append them back to the original dataframe

[100]:
```python
# There are about 2200 movies with missing budgets. Let's filter those out into␣
 ↪separate dataframe
df_0_budget = df[df.budget==0].reset_index(drop=True)

df = df[df.budget!=0].reset_index(drop=True)
```

```
# Check 0 budgets again
df.budget[df.budget==0].count()
```

[100]: 0

[101]:
```
df.shape
```

[101]: (4638, 15)

[102]:
```
# Checking data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4638 entries, 0 to 4637
Data columns (total 15 columns):
budget      4638 non-null float64
company     4638 non-null object
country     4638 non-null object
director    4638 non-null object
genre       4638 non-null object
gross       4638 non-null float64
name        4638 non-null object
rating      4638 non-null object
released    4638 non-null object
runtime     4638 non-null int64
score       4638 non-null float64
star        4638 non-null object
votes       4638 non-null int64
writer      4638 non-null object
year        4638 non-null int64
dtypes: float64(3), int64(3), object(9)
memory usage: 543.6+ KB
```

It seems like all the columns are homogenous in data type.'released' column had type String object, although it seems convertible to datetime object. To make this column ready and useful for future analysis, we'll have to split it into year, month and day columns. We can do this by first converting it to pandas' datetime object and use its attributes to extract year, month, and day

[103]:
```
df.released = pd.to_datetime(df.released) # convert to date-time type object

for i in range(len(df)):
   df.loc[i, 'release_year'] = df.released[i].year
   df.loc[i, 'release_month'] = df.released[i].month
   df.loc[i, 'release_day'] = df.released[i].day

# Now we can drop 'released' column
df = df.drop('released', axis=1)

# Check head of the dataframe
df.head()
```

[103]:
```
        budget                              company country         director  \
0   8000000.0          Columbia Pictures Corporation     USA       Rob Reiner
1   6000000.0                     Paramount Pictures     USA      John Hughes
2  15000000.0                     Paramount Pictures     USA       Tony Scott
3  18500000.0  Twentieth Century Fox Film Corporation     USA    James Cameron
4   9000000.0                   Walt Disney Pictures     USA   Randal Kleiser

       genre        gross                      name rating  runtime  score  \
0  Adventure   52287414.0               Stand by Me      R       89    8.1
1     Comedy   70136369.0  Ferris Bueller's Day Off  PG-13      103    7.8
2     Action  179800601.0                   Top Gun     PG      110    6.9
3     Action   85160248.0                    Aliens      R      137    8.4
4  Adventure   18564613.0   Flight of the Navigator     PG       90    6.9

                 star   votes         writer  year  release_year  \
0         Wil Wheaton  299174   Stephen King  1986        1986.0
1  Matthew Broderick  264740    John Hughes  1986        1986.0
2          Tom Cruise  236909       Jim Cash  1986        1986.0
3    Sigourney Weaver  540152  James Cameron  1986        1986.0
4         Joey Cramer   36636  Mark H. Baker  1986        1986.0

   release_month  release_day
0            8.0         22.0
1            6.0         11.0
2            5.0         16.0
3            7.0         18.0
4            8.0          1.0
```

Also, we can see that 'year' and 'release_year' columns seem identical. They may represent different years like the year when the movie was released vs the year it was made. Let's check if they are identical in order to drop one of them if so. First, let's check the fraction of them that's equal

[104]:
```
np.sum(df.year == df.release_year)/len(df)
```

[104]:
```
0.8436826218197498
```

That is, about 84% of the movies had similar 'release_year' and 'year' values. Now let's see if the ones that aren't equal drastically differ from each other

[105]:
```
df['year_diff'] = np.abs(df.release_year - df.year)
df[df.year !=df.release_year].loc[:, ['name', 'year', 'release_year',␣
 ↪'year_diff']].sort_values('year_diff', ascending=False)\
                                                                        ␣
 ↪                .head(10)
```

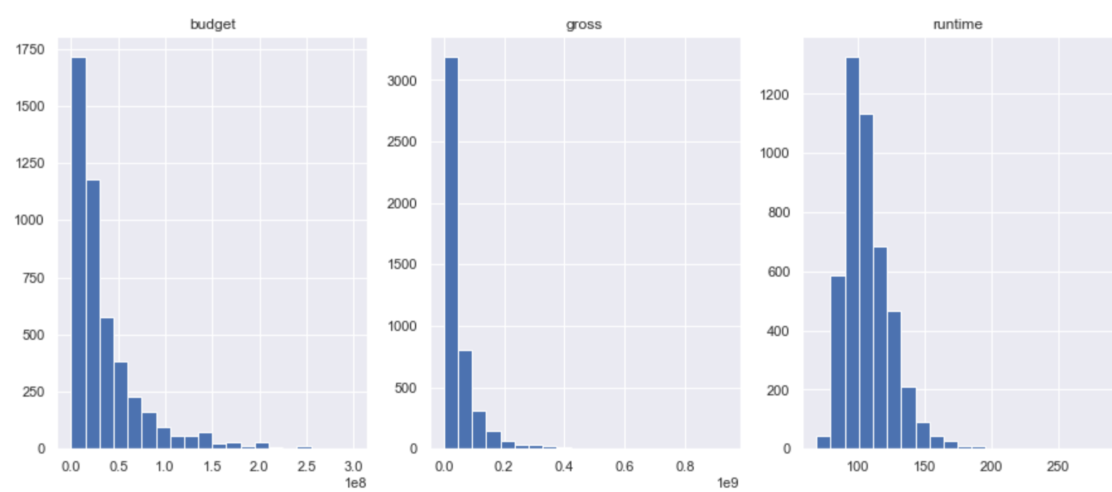
[105]:
```
                            name  year  release_year  year_diff
214    The Last Temptation of Christ  1988        2004.0       16.0
576         The Lovers on the Bridge  1991        1999.0        8.0
2571                      Eating Out  2004        2012.0        8.0
```

```
815                                    Iron Monkey  1993        2001.0        8.0
685                                   Twin Dragons  1992        1999.0        7.0
2923  Poultrygeist: Night of the Chicken Dead  2006        2012.0        6.0
1966                          The Devil's Backbone  2001        2007.0        6.0
3451                                   Tanner Hall  2009        2015.0        6.0
896                  The Legend of Drunken Master  1994        2000.0        6.0
165                                        Rampage  1987        1992.0        5.0
```

That is, there is up to 16 years difference between movies' make year and release year. Since there is significant differences between some movies' release and make years, we will not drop any of their respective columns.

Next, we will check for outliers. We are going to do this by creating histogram for each column. Since budget, gross, and runtime columns are not categorical or bounded by certain range (open ended), their histograms will be selected to detect outliers

[106]:
```
df = df.drop('year_diff', axis=1)
# to check for outliers, we are going to create histograms for each numerical␣
↪column
bins = len(df)
plt.figure(figsize=(15, 6))

plt.subplot(1, 3, 1)
plt.hist(df.budget, bins=20)
plt.title('budget')

plt.subplot(1, 3, 2)
plt.hist(df.gross, bins=20)
plt.title('gross')

plt.subplot(1, 3, 3)
plt.hist(df.runtime, bins=20)
plt.title('runtime')

plt.show()
```



It seems like most movies in the dataset had budgets below 100M, grosses below 200M, and runtimes between 75 to 150 minutes. Movies with extremely high budgets, grosses, and runtimes can be considered as outliers, therefore can be dropped. Although this may reduce the noise in the data, those extreme values, more than often, help drive the decision the most as their characteristics will be decisive in determining success (or failure) of the movie. Therefore, these extreme values will not be dropped for now

[107]:
```
df.describe(include='all')
```

[107]:
```
              budget            company country     director   genre  \
count   4.638000e+03               4638    4638         4638    4638
unique           NaN               1340      45         1892      16
top              NaN  Universal Pictures    USA  Woody Allen  Comedy
freq             NaN                265    3726           30    1310
mean    3.614560e+07                NaN     NaN          NaN     NaN
std     3.996947e+07                NaN     NaN          NaN     NaN
min     6.000000e+03                NaN     NaN          NaN     NaN
25%     1.000000e+07                NaN     NaN          NaN     NaN
50%     2.300000e+07                NaN     NaN          NaN     NaN
75%     4.600000e+07                NaN     NaN          NaN     NaN
max     3.000000e+08                NaN     NaN          NaN     NaN

               gross  name rating      runtime        score          star  \
count   4.638000e+03  4638   4638  4638.000000  4638.000000          4638
unique           NaN  4604      8          NaN          NaN          1613
top              NaN  Heat      R          NaN          NaN  Nicolas Cage
freq             NaN     2   2247          NaN          NaN            38
mean    4.607469e+07   NaN    NaN   107.595515     6.356317           NaN
std     6.629378e+07   NaN    NaN    18.022792     1.011063           NaN
min     3.090000e+02   NaN    NaN    69.000000     1.500000           NaN
25%     6.290905e+06   NaN    NaN    96.000000     5.800000           NaN
50%     2.345551e+07   NaN    NaN   104.000000     6.400000           NaN
75%     5.778243e+07   NaN    NaN   117.000000     7.100000           NaN
max     9.366622e+08   NaN    NaN   280.000000     9.300000           NaN

               votes       writer         year  release_year  release_month  \
count   4.638000e+03         4638  4638.000000   4638.000000    4638.000000
unique           NaN         2857          NaN           NaN            NaN
top              NaN  Woody Allen          NaN           NaN            NaN
freq             NaN           29          NaN           NaN            NaN
mean    9.570254e+04          NaN  2002.489435   2002.678094       6.645752
std     1.493878e+05          NaN     8.461472      8.485159       3.469357
min     1.830000e+02          NaN  1986.000000   1986.000000       1.000000
25%     1.611050e+04          NaN  1996.000000   1996.000000       4.000000
50%     4.394000e+04          NaN  2003.000000   2003.000000       7.000000
75%     1.093932e+05          NaN  2010.000000   2010.000000      10.000000
```

```
max      1.861666e+06          NaN  2016.000000   2017.000000      12.000000

        release_day
count   4638.000000
unique          NaN
top             NaN
freq            NaN
mean      16.085166
std        8.527333
min        1.000000
25%        9.000000
50%       16.000000
75%       23.000000
max       31.000000
```

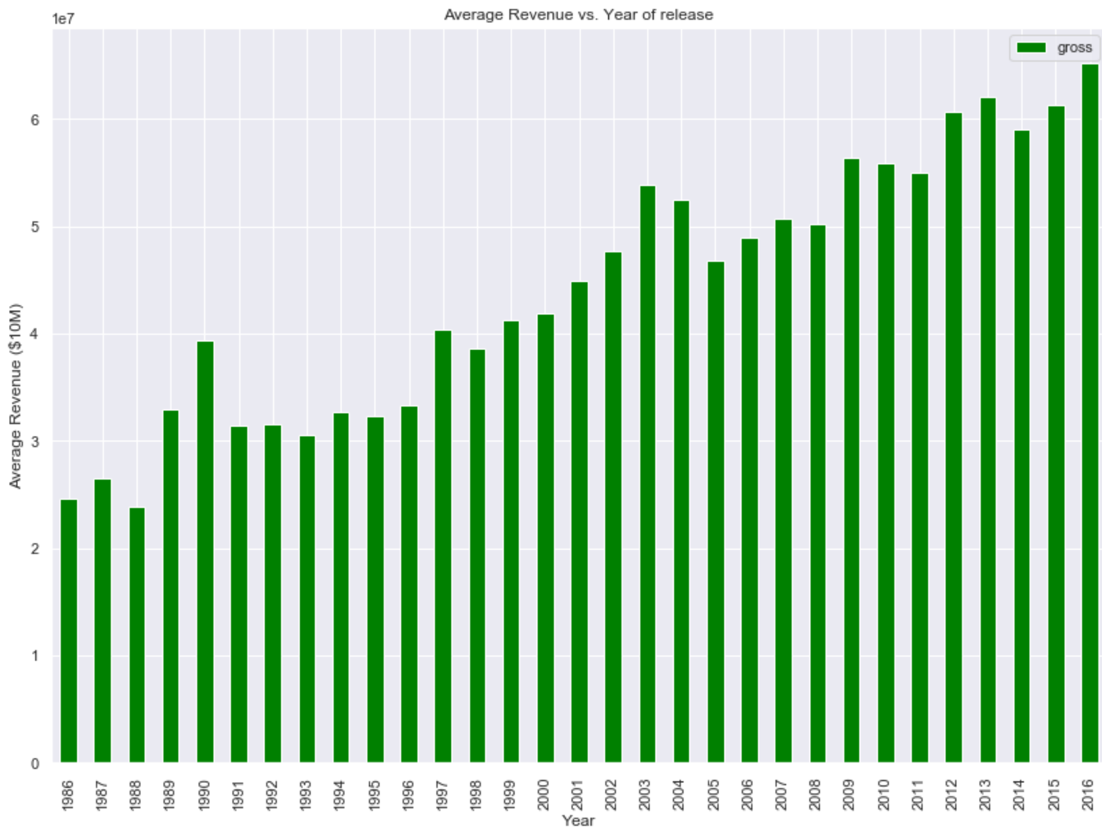
We notice that the mean of the budget and its standard deviation have increased since the last time we checked prior to data cleaning. This is because we have dropped movies with missing budget values. We will return those movies into our datasetafter estimating their budgets

Next, we will explore data's different feature columns and their relations with the target variable, revenue.

5 Data Exploration

Now let's explore any trends between features in the dataset. First task, let's see if movies' revenues have increased with time of release. To do this, we create a scatter plot of release year vs average revenue in that year

[108]:
```
mean_year = df.groupby('year').mean().reset_index()
import seaborn as sns
sns.set()
_=mean_year.plot.bar(x='year', y='gross', color='green', figsize=(14, 10))
_=plt.xlabel('Year')
_=plt.ylabel('Average Revenue ($10M)')
_=plt.title('Average Revenue vs. Year of release')
```



This graph shows a general steady incline in movies' mean revenues over the years from 1986 to 2016. This result is both interesting and surprising, why has movies' revenues increased steadily?

Although this shows a general increase in movies' profits over time, it is not stable and reliable to predict revenues based on year of release only. It does show, however, that there is more turnout to movies or that movie makers had better experience making successful movies.

Next, we will investigate relation between string columns and revenue. We can do this by creating a number of bar plots for every column where each bar represents the category of the feature. Before creating bar plots of string columns, we need to know how many categories are there in each column in order to check if it is feasible to plot that particular column as bar plot (i.e. too many categories makes infeasible bar plots)

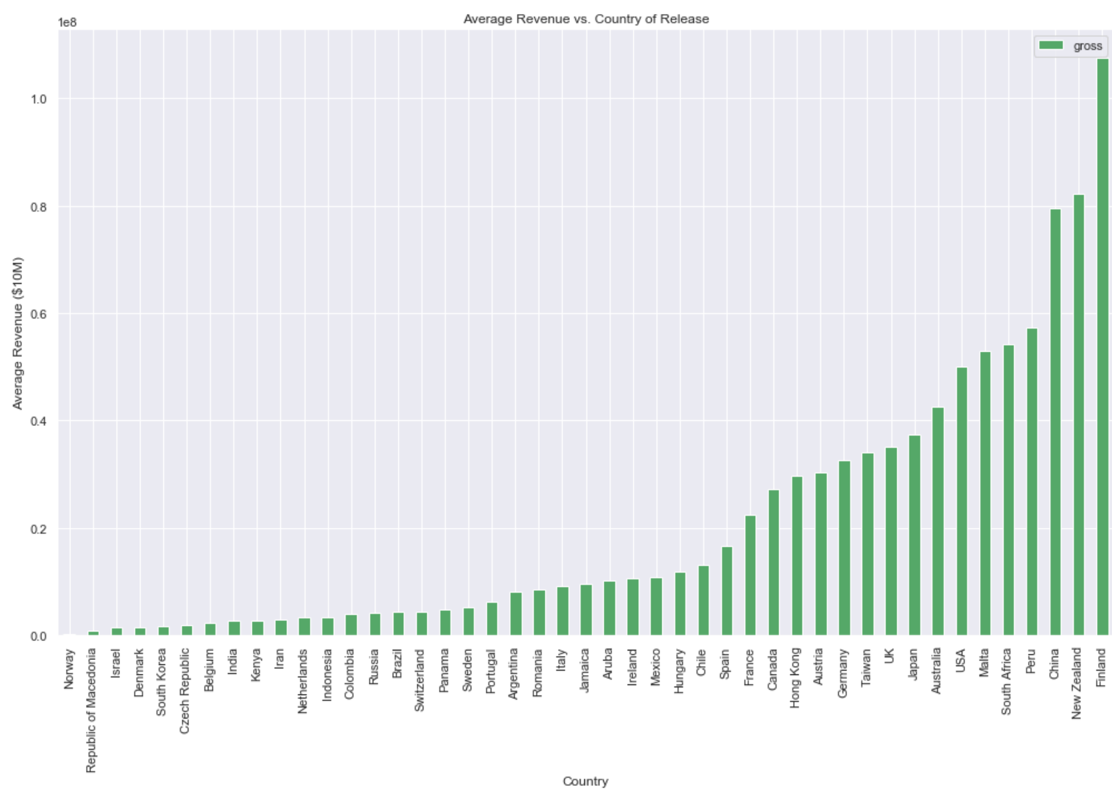
[109]:
```
df_strings = df.loc[:, ['company', 'country', 'director', 'genre', 'name',␣
 ↪'rating', 'star', 'writer']]
catgs = {}
for i in list(df_strings.columns):
    catgs[i] = df_strings[i].nunique()

catgs
```

[109]:
```
{'company': 1340,
 'country': 45,
```

```
'director': 1892,
'genre': 16,
'name': 4604,
'rating': 8,
'star': 1613,
'writer': 2857}
```

We can see that company, director, name, star, and writer of the movie columns have way too many categories so can't be represented by bar plots feasibly, otherwise, the rest will be bar-plotted against revenue

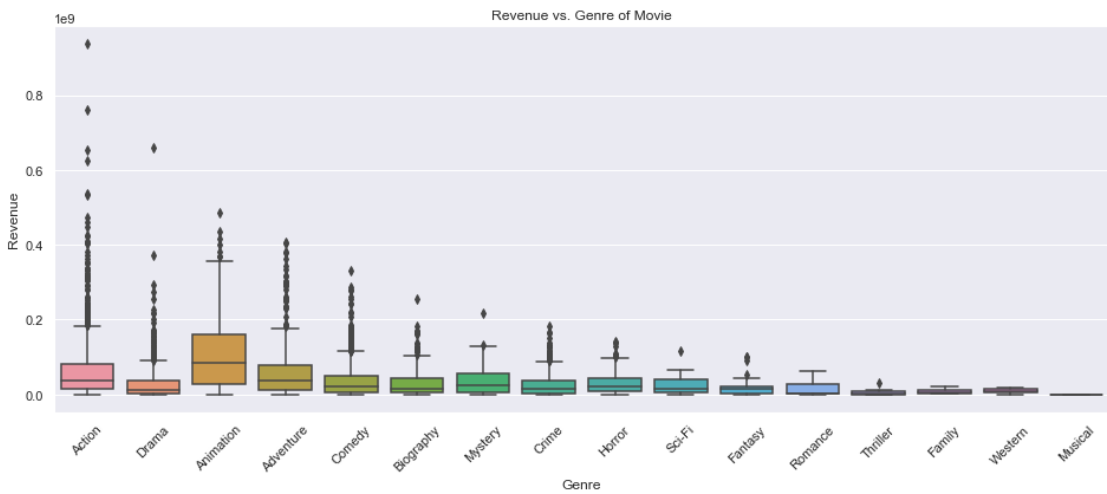
[110]:
```
_=df.groupby('country').mean().reset_index().sort_values('gross').plot.
↪bar(x='country', y='gross', figsize=(17,10), color='g')
_=plt.xlabel('Country')
_=plt.ylabel('Average Revenue ($10M)')
_=plt.title('Average Revenue vs. Country of Release')
```



It looks like Finland, New Zealand, and China had the highest average revenues for movies they released over the past 20 years. Surprisingly, most of the other countries had, on average, lost on unsuccessful movies. This is a good indicator that movies released in those countries will most likely be successful

Next, we continue to explore how genre of movie influences its revenues. We will use a box plot as it is less prone to outliers in the genre which may have very high or low gross movies

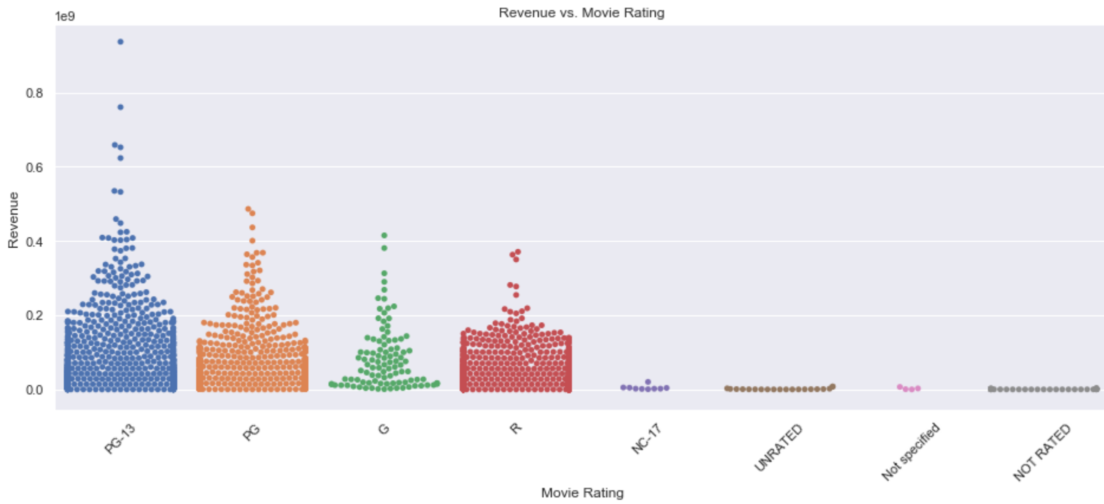
[111]:
```
# Now let's check what genre of movies had the greatest revenue
plt.figure(figsize=(16, 6))
_=sns.boxplot(x='genre', y='gross', data=df.sort_values('gross',␣
↪ascending=False))
_=plt.xlabel('Genre')
_=plt.ylabel('Revenue')
_=plt.xticks(rotation=45)
_=plt.title('Revenue vs. Genre of Movie')
```



This boxplot shows that genre is not decisive in determining revenue of the movie. Nevertheless, Action and Drama movies seem to be able to make more revenues than other genres like Westerns and Musicals

Coming up, we'll use a swarm plot to visualize relation between movie rating and revenue. Swarm plots make it easier to see how data is distributed

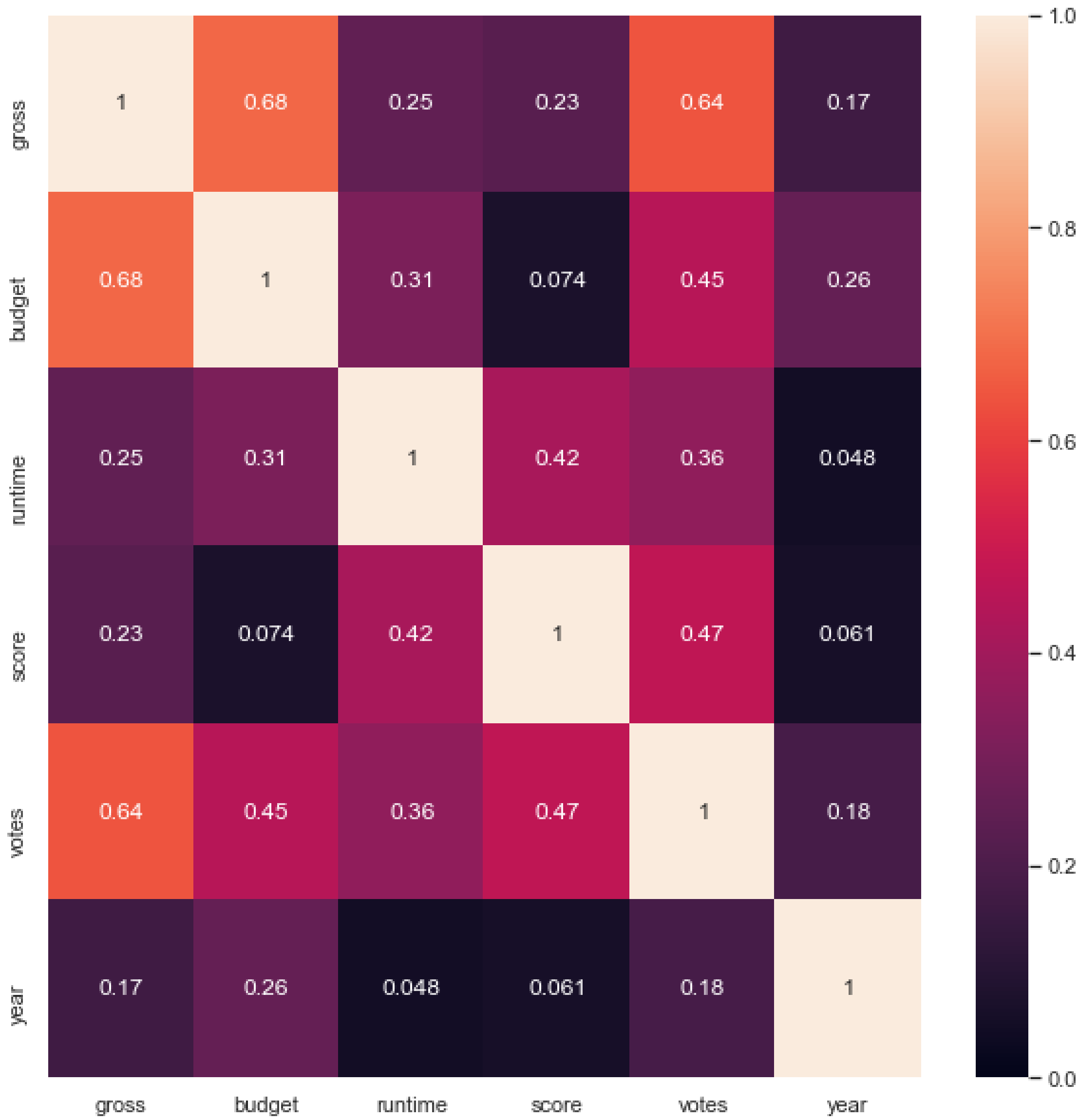
[48]:
```
# Finally, let's check how movie rating influences its revenue, if it does.
plt.figure(figsize=(16, 6))
_=sns.swarmplot(x='rating', y='gross', data=df.sort_values('gross',␣
↪ascending=False))
_=plt.xlabel('Movie Rating')
_=plt.ylabel('Revenue')
_=plt.xticks(rotation=45)
_=plt.title('Revenue vs. Movie Rating')
```



Although this result is not decisive, it shows, to a great extent, that movies with PG-13 and PG ratings tend to be able to make more revenue than those unrated or TV-MA.

After exploring several variables' relations with revenue, it is time to see which of numerical variables correlates the most with revenue. This requires finding pearson correlation between each of the columns. We will also create a heatmap to visualize correlation strength in a color spectrum

[112]:
```
plt.figure(figsize=(10,10))
ax = sns.heatmap(df.loc[:, ['gross', 'budget', 'runtime', 'score', 'votes',␣
 ↪'year']].corr(), vmin=0, vmax=1, annot=True)
```

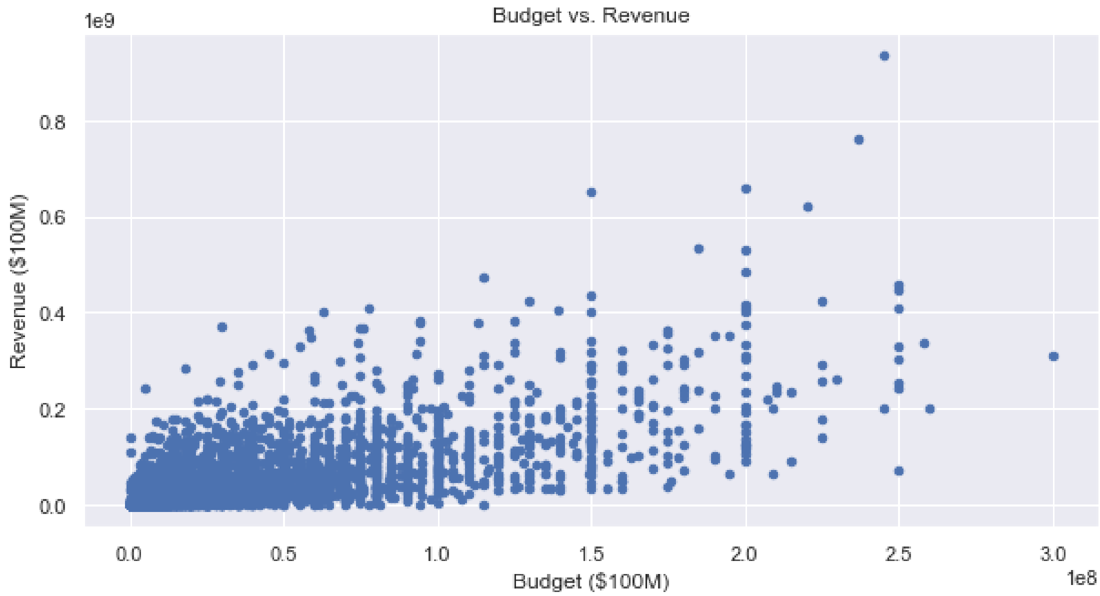


From this, we can see that none of the numerical columns decisively correlate with revenue, although budget and number of votes show a relatively significant correlation. The strong correlation between revenue and budget of the movie makes sense. The more the money is spent on one movie the more likely the movie will make more revenues. Next, we will explore this relationship in a scatter plot

[113]:
```
df.plot.scatter(x='budget', y='gross', figsize=(10, 5))
_=plt.xlabel('Budget ($100M)')
_=plt.ylabel('Revenue ($100M)')
_=plt.title('Budget vs. Revenue')
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
```

```
'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



This shows a roughly linear relation between the two. We are not sure if this result is a mere coincidence or there is in fact a linear relation between budget and revenue of the movie. In order to verify this assumption, we need to do a hypothesis test. A suitable hypothesis test would involve finding fraction of pearson correlations in permuted replicates that are at least as extreme as observed one (0.68) to the total number of replicates, which is p_value. If that p_value < 0.05, we reject the null hypothesis as there is statistically significant correlation

H0: No significant correlation between Budget and Revenue

H1: There is significant correlation between them

[114]:
```python
# create pearson correlation function that takes two vectors and finds␣
↪correlation between them
def pearson_r(x, y):
    corr_mat = np.corrcoef(x, y)
    return corr_mat[0,1]
# next, we calculate observed correlation
r_observed = pearson_r(df.budget, df.gross)
# instantiate permuted replicates of correlations
perm_replicates = np.empty(100000)
# next, we create permutated gross array and its correlation matrix with gross␣
↪for 100,000 times
for i in range(100000):
    budget_permuted = np.random.permutation(df.budget)
    perm_replicates[i] = pearson_r(budget_permuted, df.gross)

p_value = np.sum(perm_replicates>=r_observed)/len(perm_replicates)
print("p_value = ", p_value)
```

```
p_value =  0.0
```

From the result above, the p_value is less than < 0.05 and therefore, null hypothesis can be rejected in favor of the alternative. In this case, we know that there is a statistically significant correlation between budget and revenue. This relationship could be useful for later analysis as it may help build revenue-predictive models based on budget

From the visualizations made between many feature columns and the target variable, revenue, we have seen that some features strongly correlate with the target while others do not. For those that did correlate with the target variable, most notably the budget of movie, we have created a hypothesis and tested that hypothesis which resulted in a very small p_value and hence concluded that there is statistically significant correlation between budget and revenue.

Next steps include encoding string columns and applying dimensionality reduction to cut down the number of columns such that machine learning algorithms will be most useful and accurate down the line. After creating ML models, we will test their accuracies and determine if there needs to be additional data to support the regression task. If needed, those will be joined on the movie name and will include information like movie franchise, remakes, and publicity.

6 In-Depth Analysis

In this section, we will try to create a model that is able to predict revenue of the movie with high precision. First, we will try out different models, with their standard out-of-the-box parameters, to check which performs best. The best performer will then be fine tuned to achieve best results.

Before going forward with creating models to predict gross, from the result in previous section, we saw that gross strongly correlates with budget. We could use this result to estimate budgets for movies that had unknown budgets filtered out in the wrangling section. We could add those in the dataframe to create more data and therefore help models predict gross more accurately. We will use the result from previous section to create a linear model (ridge with L2 regularization) to predict budget from gross.

[50]:
```python
from sklearn.linear_model import Ridge

ridge = Ridge()
ridge.fit(df.gross.values.reshape(-1, 1), df.budget.values.reshape(-1, 1))
df_0_budget.budget = ridge.predict(df_0_budget.gross.values.reshape(-1, 1))

df_0_budget.head()
```

[50]:
```
         budget                      company country             director  \
0  3.394105e+07              TriStar Pictures     USA          John Badham
1  2.018785e+07          Neue Constantin Film   Italy   Jean-Jacques Annaud
2  2.716049e+07              TriStar Pictures     USA       Sidney J. Furie
3  1.807648e+07                       Gaumont  France  Jean-Jacques Beineix
4  6.444770e+07  Columbia Pictures Corporation     USA      John G. Avildsen

    genre       gross                   name   rating    released  \
0  Comedy  40697761.0          Short Circuit       PG  1986-05-09
1   Crime   7153487.0  The Name of the Rose        R  1986-09-24
2  Action  24159872.0             Iron Eagle   PG-13  1986-01-17
```

```
3   Drama     2003822.0              Betty Blue  Not specified  1986-11-07
4  Action   115103979.0  The Karate Kid Part II             PG  1986-06-20

   runtime  score                  star  votes             writer  year
0       98    6.6           Ally Sheedy  47068        S.S. Wilson  1986
1      130    7.8          Sean Connery  86991        Umberto Eco  1986
2      117    5.3     Louis Gossett Jr.  11304  Kevin Alyn Elders  1986
3      120    7.4  Jean-Hugues Anglade  14562     Philippe Djian  1986
4      113    5.9            Pat Morita  58370  Robert Mark Kamen  1986
```

Next, we convert release date to separate columns, as we done before, then we append this dataframe to the original dataframe

[51]:
```
df_0_budget.released = pd.to_datetime(df_0_budget.released)
for i in range(len(df_0_budget)):
   df_0_budget.loc[i, 'release_year'] = df_0_budget.released[i].year
   df_0_budget.loc[i, 'release_month'] = df_0_budget.released[i].month
   df_0_budget.loc[i, 'release_day'] = df_0_budget.released[i].day

# Now we can drop 'released' column
df_0_budget = df_0_budget.drop('released', axis=1)

df = df.append(df_0_budget) # check the shape of the dataframe
df.shape
```

[51]:
```
(6820, 17)
```

Great, now our dataframe has same size as the original dataset with better budget estimations. Now we can start creating gross-predictive models

First, we need to encode the categorical data to input them in any model created. We will numerically encode these categories (0 to N-number of categories) using pandas' factorize function. The string columns we have are the company, country, director, genre, name, rating, star, and writer of the movie. Name of the movie, however, does not mean much in the decision process. For now, let's drop it out of the dataframe. We will use it as merge-on column if we need additional information about each movie.

[54]:
```
movie_names = df.pop('name') # remove it for now, use it later if needed
```

[55]:
```
for i in np.array(['company', 'country', 'director', 'genre', 'rating', 'star',␣
↪'writer']):
  df.loc[:, i], labels = pd.factorize(df.loc[:, i])

df.head() # check the dataframe
```

[55]:
```
      budget  company  country  director  genre        gross  rating  \
0  8000000.0        0        0         0      0   52287414.0       0
1  6000000.0        1        0         1      1   70136369.0       1
2 15000000.0        1        0         2      2  179800601.0       2
3 18500000.0        2        0         3      2   85160248.0       0
4  9000000.0        3        0         4      0   18564613.0       2
```

```
   runtime  score  star   votes  writer  year  release_year  release_month  \
0       89    8.1     0  299174       0  1986        1986.0            8.0
1      103    7.8     1  264740       1  1986        1986.0            6.0
2      110    6.9     2  236909       2  1986        1986.0            5.0
3      137    8.4     3  540152       3  1986        1986.0            7.0
4       90    6.9     4   36636       4  1986        1986.0            8.0

   release_day
0         22.0
1         11.0
2         16.0
3         18.0
4          1.0
```

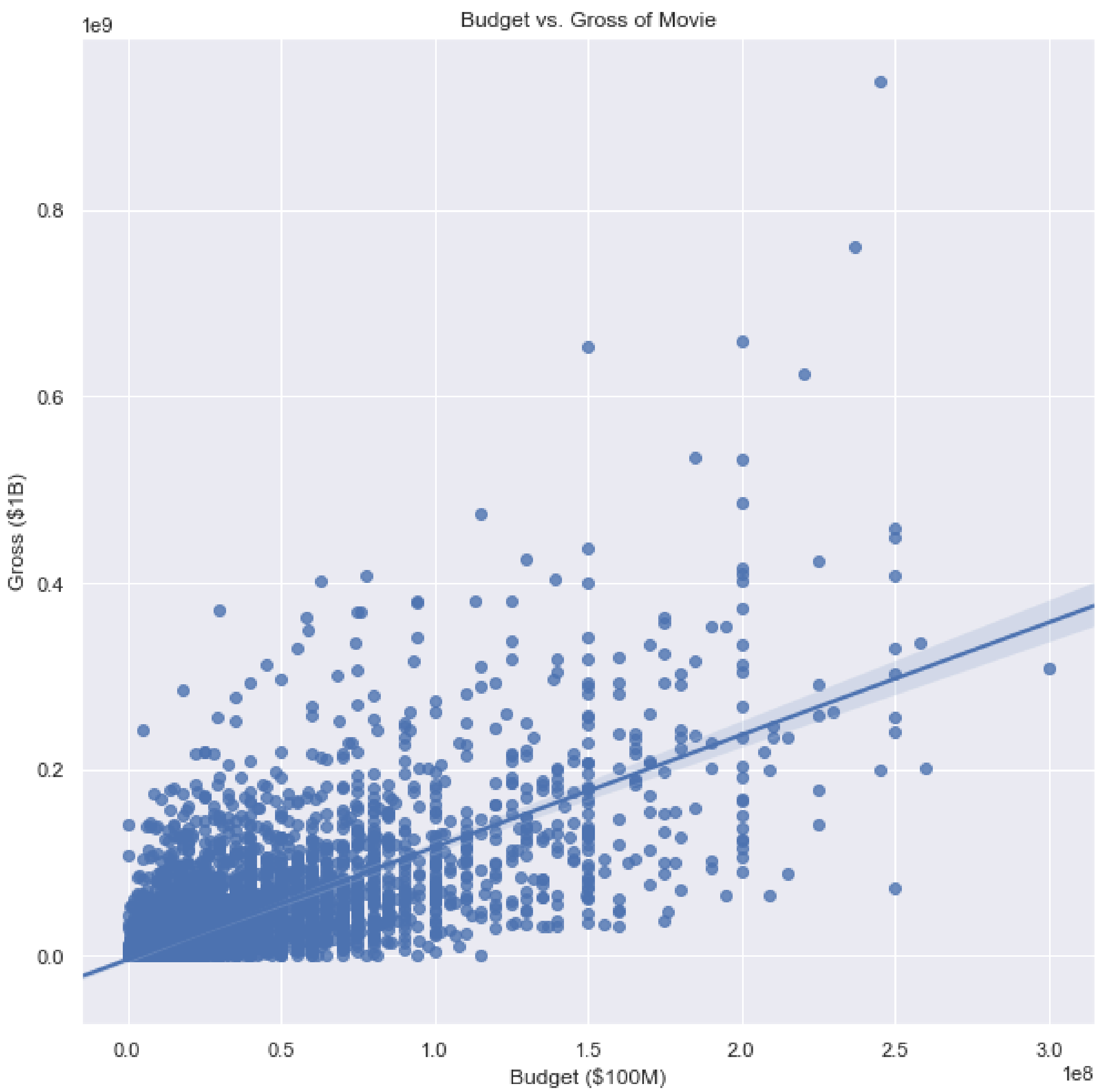
Now our dataframe is all numerical type and can be used in ML algorithms, which is what we will do next. Before that, we need to split the data to train and test sets. We will choose 30% for test set and specify random state for reproducibility

[57]:
```
from sklearn.model_selection import train_test_split

# evaluation metric
from sklearn.metrics import mean_squared_error as MSE
X = df.drop(['gross'], axis=1) # we want to drop the gross as it represents the␣
↪target variable
y = df.gross # gross/revenue as target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=20) # split to 30% test, rest is train
```

Based on our hypothesis test from previous section, the first model we will create is a linear regression model to estimate gross based on budget only. The model will be using L2 regularization (Ridge regression) as it is less prone to outliers. We will instantiate, train, and test the model and then evaluate its performance using root-mean-squared-error (RMSE). Let's plot how a line would look like between the two

[58]:
```
plt.figure(figsize=(10, 10))
sns.regplot('budget', 'gross', data=df)
plt.xlabel('Budget ($100M)')
plt.ylabel('Gross ($1B)')
plt.title('Budget vs. Gross of Movie')
plt.show()
```

```
C:\Users\ali95\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713:
FutureWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Now, let's create the model

[59]:
```
ridge = Ridge()
ridge.fit(X_train.budget.values.reshape(-1, 1), y_train.values.reshape(-1, 1))
y_pred = ridge.predict(X_test.budget.values.reshape(-1, 1))
print("RMSE result for Budget-Gross Regression = ", np.sqrt(MSE(y_test,␣
 ↪y_pred)))
```

```
RMSE result for Budget-Gross Regression =  41703831.25660343
```

The resulting RMSE is about 42E6. That is, for every gross prediction, there is, on average, about 42M dollars in tolerance about the true revenue. Although this is a powerful result, we will investigate other models that will account for **all** other variables in the decision to see if it is possible to narrow down this tolerance

Here, we will create a function that will instantiate, train, and test different algorithms and return their results in a dataframe. Here, we will use 5 different regressors; random forest, Adaptive Boosting, Gradient Boosting, K-Nearest Neighbor, and Ridge regressors. The reasons why we chose those algorithms are discussed in the paper

[60]:
```
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
 ↪GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor

def best_classifier(X_train, y_train, X_test, y_test):

   # Random Forest
    random_forest = RandomForestRegressor(n_estimators=100)
    random_forest.fit(X_train, y_train)
    train_acc_rf = str(round(random_forest.score(X_train, y_train)*100, 2))+"%"
    test_acc_rf = str(round(random_forest.score(X_test, y_test) * 100, 2))+"%"

    dt = DecisionTreeRegressor()
    # Adaptive Boosting
    abr = AdaBoostRegressor(base_estimator=dt)
    abr.fit(X_train, y_train)
    train_acc_abr = str(round(abr.score(X_train, y_train)*100, 2))+"%"
    test_acc_abr = str(round(abr.score(X_test, y_test)*100, 2))+"%"

    # Gradient Boosting
    gbr = GradientBoostingRegressor()
    gbr.fit(X_train, y_train)
    train_acc_gbr = str(round(gbr.score(X_train, y_train)*100, 2))+"%"
    test_acc_gbr = str(round(gbr.score(X_test, y_test)*100, 2))+"%"

    # k-Nearest Neighbors
    knn = KNeighborsRegressor(n_neighbors = 3)
    knn.fit(X_train, y_train)
    train_acc_knn = str(round(knn.score(X_train, y_train)*100, 2))+"%"
    test_acc_knn = str(round(knn.score(X_test, y_test)*100, 2))+"%"

    # Ridge
    ridge = Ridge()
    ridge.fit(X_train, y_train)
    train_acc_ridge = str(round(ridge.score(X_train, y_train)*100, 2))+"%"
    test_acc_ridge = str(round(ridge.score(X_test, y_test)*100, 2))+"%"


    results = pd.DataFrame({
    'Model': ['KNN', 'Random Forest', 'Ridge', 'Gradient Boosting', 'AdaBoost'],
    'Train_Score': [train_acc_knn, train_acc_rf, train_acc_ridge,␣
 ↪train_acc_gbr, train_acc_abr],
```

```
    'Test_Score': [test_acc_knn, test_acc_rf, test_acc_ridge, test_acc_gbr,␣
↪test_acc_abr]})

    results=results.set_index('Model')

    return (results)

results_mat = best_classifier(X_train, y_train, X_test, y_test)
results_mat.sort_values('Test_Score', ascending=False)
```

[60]:

```
                  Train_Score Test_Score
Model
AdaBoost               99.97%     77.26%
Random Forest          95.95%     77.14%
Gradient Boosting      84.47%     77.01%
Ridge                  63.49%     67.84%
KNN                    78.08%     60.16%
```

After running the function with the training and testing data and printing the results matrix, we see that AdaBoost followed by random forest and Gradient Boosting regressors perform best with similar accuracies on test data compared to Ridge and KNN models. It is worth mentioning, however, that Gradient Boosting had significantly lower train accuracy compared to random forest and AdaBoost. Random forest and adaBoost regressors, for example, had about 10% and 14% higher train accuracies, respectively. This indicates that those models are overfitting the training set and need to be simplified to account for variance in the data. For this reason, we will select Gradient Boosting regressor.

Next we will proceed with Gradient Boosting regressor. We will fine-tune this model by creating a grid of parameters and possible values using sklearn's GridSearchCV. The parameters we are going to tune are learning rate, number of estimators, minimum number of samples/leaf, and maximum depth of base tree

[82]:

```
from sklearn.model_selection import GridSearchCV
gbr = GradientBoostingRegressor()
params = {'learning_rate':[0.01, 0.1, 1], 'n_estimators':[100, 200, 300, 400,␣
↪500], 'min_samples_leaf':[1, 2, 3, 4, 5]}

model_cv = GridSearchCV(gbr, param_grid=params, cv=4, n_jobs=-1)
model_cv.fit(X_train, y_train)

print("Best model params ", model_cv.best_params_)
print("Best model score = ", model_cv.best_score_)
```

```
Best model params  {'learning_rate': 0.1, 'min_samples_leaf': 5, 'n_estimators':
500}
Best model score =  0.7381769551356597
```

After creating a grid search, we have found the optimal parameters for the model to perform best. Let's plug those in and find the test accuracy

[93]:
```
model_tuned = GradientBoostingRegressor(learning_rate=0.1, min_samples_leaf =␣
↪5, n_estimators = 500, max_depth=4)
model_tuned.fit(X_train, y_train)
print("Tuned model test accuracy = ", model_tuned.score(X_test, y_test))
```

```
Tuned model test accuracy =  0.799053629400203
```

After tuning the model and testing its accuracy, it's more appealing to quantify error in a way that makes sense. Since our task is regression, we will use root mean squared error (RMSE) as our evaluation metric as it's directly interpretable to measurement units, which is dollars in gross in this case

[94]:
```
y_pred = model_tuned.predict(X_test)

print("Root Mean Squared Error (RMSE) = ", np.sqrt(MSE(y_test, y_pred)))
```

```
Root Mean Squared Error (RMSE) =  26433581.164229058
```

We see that RMSE result is about 26E6. That is, for every movie's gross prediction, there is, on average, $26 million tolerance about the true gross/revenue. This is a huge improvement in tolerance since the budget-gross regression model. Although this amount seems high as movie companies gross estimation may be off by that amount, nevertheless, this amount makes only small portion of most of nowadays movies' budgets and estimating movie gross with that tolerance may still be of great benefit to evaluate a movie even before its make.

7 Conclusion

from the previous section, we can draw the following conclusions: * Movie gross is strongly dependent on its allocated budget * Gradient boosting regressor, unlike random forest and Adaptive Boosting regressors, shows high reluctancy to overfitting * Considering movie information like its genre, release year, and so helps nail down the gross prediction * Fine tuning the model will only slightly increase its accuracy. * Other information about movie like its target audience, publicity, remakes, and such could help improve regression