Movie Revenue Prediction

June 30, 2021

Group Member: Jeffrey Zhang, Oliver Liu, Zhe Zhou

[1]:
```
import pandas as pd
import numpy as np
import matplotlib as plt
import matplotlib.pyplot as pyplt
import seaborn as sns
import datetime
import calendar
import statistics
```

EDA

- Some of the columns contain lists and dictionaries. Extract information you need and reformat them. – Zhe Zhou

[2]:
```
# Read the data from the file, and create a DataFrame object.
raw_data_movies = pd.read_csv("tmdb_5000_movies.csv")
```

[3]:
```
# Reformat the columns contain dictionaries as a string list.
raw_data_movies["genres"] = raw_data_movies["genres"].apply(lambda x :␣
↪[i["name"] for i in eval(x)])
raw_data_movies["keywords"] = raw_data_movies["keywords"].apply(lambda x :␣
↪[i["name"] for i in eval(x)])
raw_data_movies["production_companies"] =␣
↪raw_data_movies["production_companies"].apply(lambda x : [i["name"] for i in␣
↪eval(x)])
raw_data_movies["production_countries"] =␣
↪raw_data_movies["production_countries"].apply(lambda x : [i["name"] for i in␣
↪eval(x)])
raw_data_movies["spoken_languages"] = raw_data_movies["spoken_languages"].
↪apply(lambda x : [i["name"] for i in eval(x)])
```

[4]:
```
# Read the data from the file, and create a DataFrame object.
raw_data_credits = pd.read_csv("tmdb_5000_credits.csv")
```

[5]:
```
# Reformat the columns contain dictionaries as a string list.
raw_data_credits["cast"] = raw_data_credits["cast"].apply(lambda x : [i["name"]␣
↪for i in eval(x)])
```

```
raw_data_credits["crew"] = raw_data_credits["crew"].apply(lambda x : [i["job"]␣
↪+ " : " + i["name"] for i in eval(x)])
```

[6]:
```
# Merge two datasets base on the movies' id number, and drop the duplicated␣
↪columns.
raw_data = pd.merge(raw_data_movies, raw_data_credits.drop("title", 1), left_on␣
↪= "id", right_on = "movie_id").drop("movie_id", 1)
```

- Clean the dataset, remove the outliers, before any data analysis. Explain what you did. – Zhe Zhou

[7]:
```
# Clean the dataset, and remove the outliers.
data = raw_data[(raw_data["budget"] > 0) &
                (raw_data["original_title"] is not np.nan) &
                (raw_data["popularity"] > 0) &
                (raw_data["production_companies"].apply(len) != 0) &
                (raw_data["production_countries"].apply(len) != 0) &
                (raw_data["release_date"] is not np.nan) &
                (raw_data["revenue"] > 0) &
                (raw_data["runtime"] > 0) &
                (raw_data["cast"].apply(len) != 0) &
                (raw_data["crew"].apply(len) != 0)]
```

In the process of cleaning the data, some outliers that are caused by artifacts have to be removed first. The purpose of this project is to build the model of predicting the revenue of movies, so the values of budget and revenue are not supposed to be zero. Also, the runtime of the movies cannot be zero because it does not make sense. In addition, the columns, “original_title”, “cast”, and “crew”, are necessary since they demonstrate the convincingness of the data. Furthermore, in the other columns, “production_companies” and “production_countries”, all these data are required in the model that we are going to build. And now, we are able to begin our data analysis.

[8]:
```
data.describe()
```

[8]:
```
             budget             id   popularity       revenue      runtime  \
count  3.183000e+03    3183.000000  3183.000000  3.183000e+03  3183.000000
mean   4.113039e+07   44878.875589    29.415936  1.229086e+08   110.859881
std    4.450600e+07   75046.011568    36.283411  1.871212e+08    20.991509
min    1.000000e+00       5.000000     0.037073  5.000000e+00    41.000000
25%    1.100000e+07    4884.500000    10.812450  1.770142e+07    96.000000
50%    2.600000e+07   11361.000000    20.786616  5.693230e+07   107.000000
75%    5.500000e+07   45038.500000    37.689512  1.487174e+08   121.000000
max    3.800000e+08  417859.000000   875.581305  2.787965e+09   338.000000

       vote_average   vote_count
count   3183.000000  3183.000000
mean       6.315112   991.026076
std        0.868237  1419.826830
min        0.000000     0.000000
```

```
25%          5.800000     189.000000
50%          6.300000     484.000000
75%          6.900000    1161.000000
max          8.500000   13752.000000
```

- Count the number of movies released by day of week, month and year, are there any patterns that you observe? – Oliver Liu

[9]:
```
data['release_date']
```

[9]:
```
0       2009-12-10
1       2007-05-19
2       2015-10-26
3       2012-07-16
4       2012-03-07
           …
4773    1994-09-13
4788    1972-03-12
4792    1997-11-06
4796    2004-10-08
4798    1992-09-04
Name: release_date, Length: 3183, dtype: object
```

[10]:
```
days = []
for date in data['release_date']:
    day = calendar.day_name[datetime.datetime.strptime(date, '%Y-%m-%d').
 ↪weekday()]
    days.append(day)
data["release_day_of_week"] = days
data
groupby_day = data.groupby('release_day_of_week').budget.count()
print(groupby_day.sort_values())
```

```
release_day_of_week
Sunday        112
Saturday      129
Monday        157
Tuesday       223
Wednesday     593
Thursday      665
Friday       1304
Name: budget, dtype: int64
```

```
<ipython-input-10-4c2d22df4d15>:5: 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#returning-a-view-versus-a-copy
```

```
data["release_day_of_week"] = days
```

- What are the movie genre trend shifting patterns that you can observe from the dataset? – Jeffrey Zhang

[11]:
```
# Gets all genres in the dataset
unique_genre = {genre for l in data["genres"] for genre in l}
unique_genre
```

[11]:
```
{'Action',
 'Adventure',
 'Animation',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'Foreign',
 'History',
 'Horror',
 'Music',
 'Mystery',
 'Romance',
 'Science Fiction',
 'Thriller',
 'War',
 'Western'}
```

[12]:
```
# Gets the popularity of all genres including repeats different genres
all_info = {}
for ug in unique_genre:
    list = []
    for l in range (0,len(data["popularity"])):
        nextList = data["genres"].get(l)
        if (nextList is not None and ug in nextList):
            list.append(data["popularity"].get(l))
    all_info[ug] = list
```

[13]:
```
# Removes any genre with no popularity
new_all_info = {key:val for key, val in all_info.items() if val}
```

[14]:
```
all_info = new_all_info
genres = [*all_info]
genres
```

[14]:
```
['Action',
 'Animation',
```

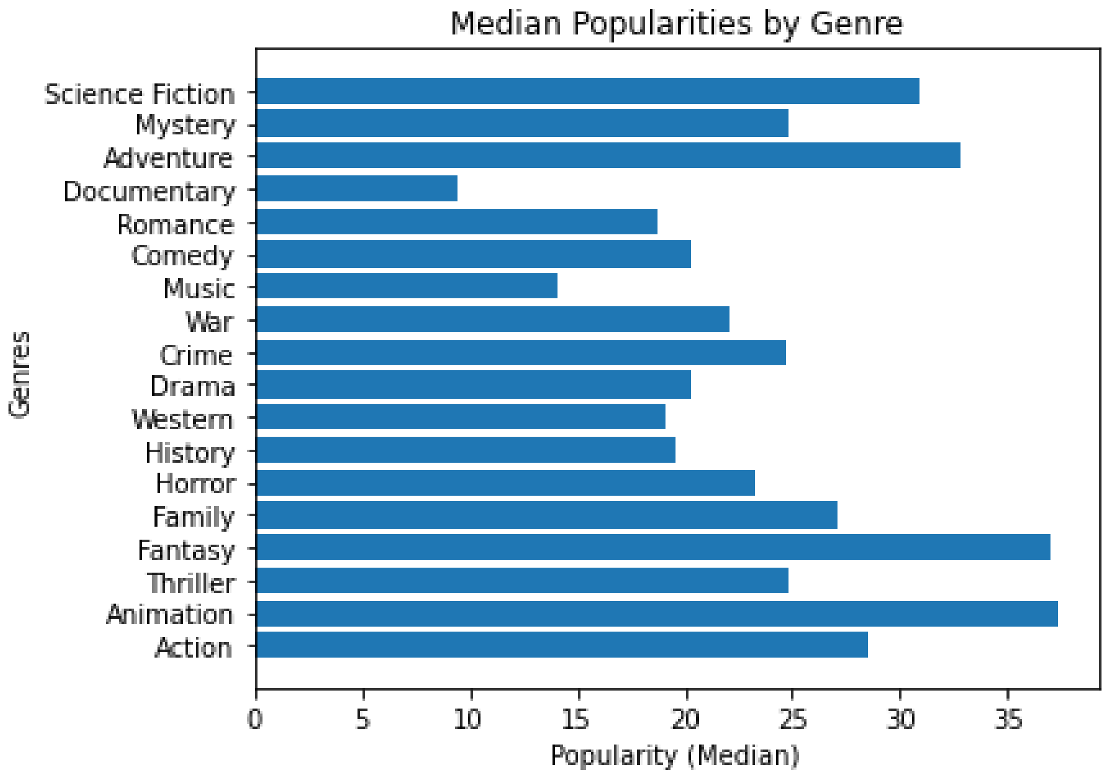
```
 'Thriller',
 'Fantasy',
 'Family',
 'Horror',
 'History',
 'Western',
 'Drama',
 'Crime',
 'War',
 'Music',
 'Comedy',
 'Romance',
 'Documentary',
 'Adventure',
 'Mystery',
 'Science Fiction']
```

[15]:
```
# Uses previous dictionary to get medians and means for each genre
medians = {}
means = {}
```

[16]:
```
for g in genres:
    list = all_info.get(g)
    if(list):
        medians[g] = statistics.median(list)
        means[g] = statistics.mean(list)
median_values = [*medians.values()]
mean_values = [*means.values()]
median_val_rounded = [round(num,2) for num in median_values]
mean_val_rounded = [round(num,2) for num in mean_values]
#len(median_val_rounded)
#len(mean_val_rounded)
```

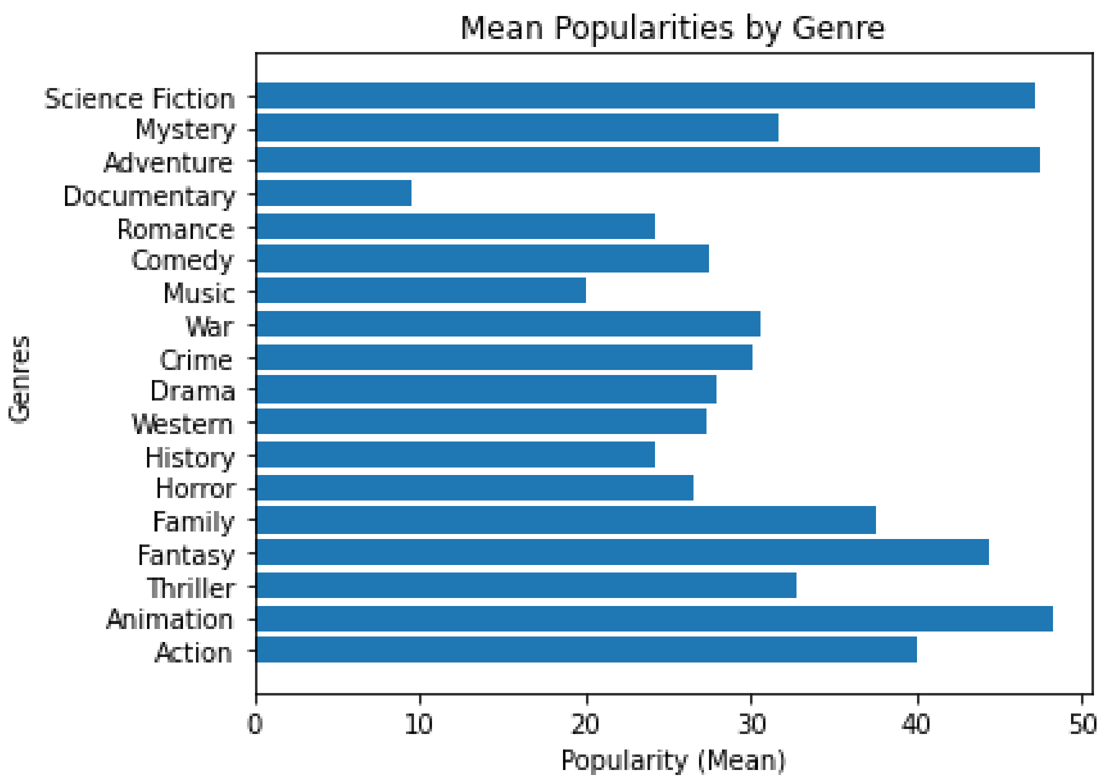
[17]:
```
# Plot data
pyplt.barh(y=genres,width=median_val_rounded)
pyplt.tight_layout()
pyplt.xlabel("Popularity (Median)")
pyplt.ylabel("Genres")
pyplt.title("Median Popularities by Genre")
```

[17]:
```
Text(0.5, 1.0, 'Median Popularities by Genre')
```



[18]:
```python
pyplt.barh(y=genres,width=mean_val_rounded)
pyplt.tight_layout()
pyplt.xlabel("Popularity (Mean)")
pyplt.ylabel("Genres")
pyplt.title("Mean Popularities by Genre")
```

[18]: Text(0.5, 1.0, 'Mean Popularities by Genre')



Via my interpretation of the question, "What are the movie genre trend shifting patterns that you can observe from the dataset?", I started by understanding what trends are which are usually the most popular object which means dictates that trend shifting would imply an object in this case our object being movie genre that is farest away from the mean and medians. To get this information, I used the dataset to find all unique genres to find the popularity means and medians for each genre. AfterwardsI used the median and means by genre to visualize the results which displays that documentaries are the movie genre that shifts the movie genre trend pattern the most since it is by far the lowest in both median and mean compared to all other moviegenres.

- What are the strongest and weakest features correlated with movie revenue? – Oliver Liu

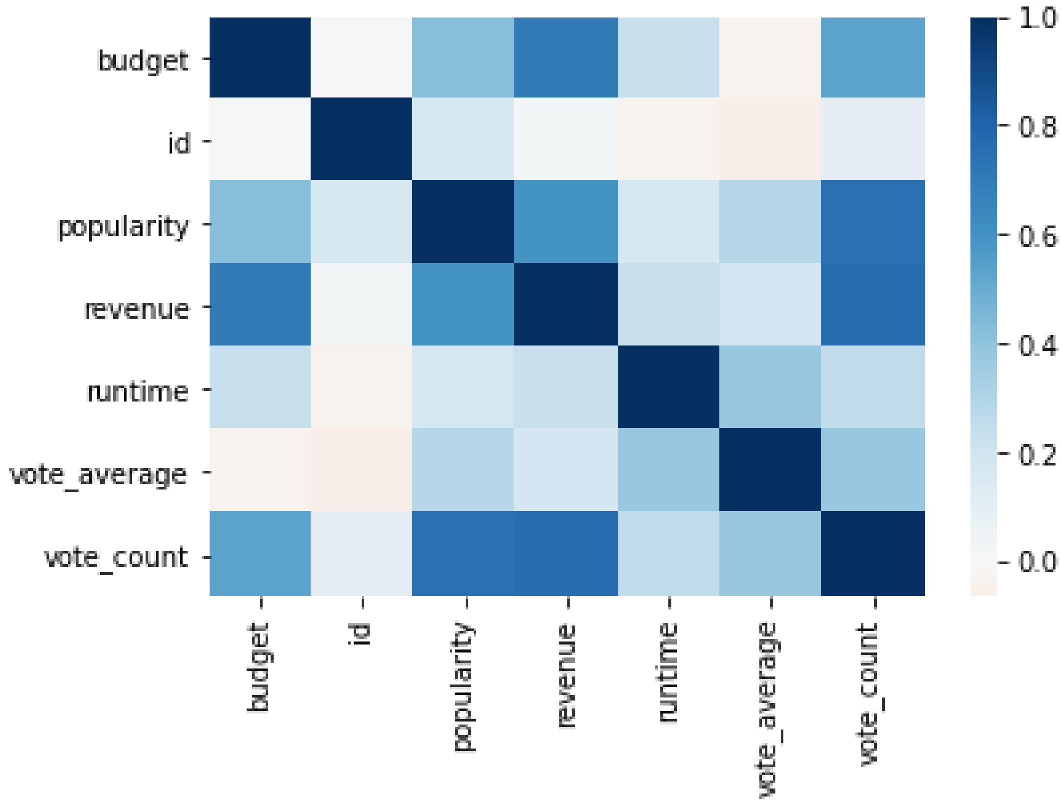
[19]:
```
data.corr()
```

[19]:
```
                budget        id  popularity   revenue   runtime  \
budget        1.000000  0.012717    0.427822  0.703984  0.226795
id            0.012717  1.000000    0.178044  0.029373 -0.033730
popularity    0.427822  0.178044    1.000000  0.599706  0.179201
revenue       0.703984  0.029373    0.599706  1.000000  0.231085
runtime       0.226795 -0.033730    0.179201  0.231085  1.000000
vote_average -0.034135 -0.064647    0.286779  0.187030  0.382346
vote_count    0.537224  0.106548    0.747323  0.754761  0.255873

              vote_average  vote_count
```

```
budget           -0.034135    0.537224
id               -0.064647    0.106548
popularity        0.286779    0.747323
revenue           0.187030    0.754761
runtime           0.382346    0.255873
vote_average      1.000000    0.379500
vote_count        0.379500    1.000000
```

[20]:
```
sns.heatmap(data.corr(), cmap='RdBu', center=0)
```

[20]: <AxesSubplot:>



[21]:
```
groupby_day_rev = data.groupby('release_day_of_week').revenue.agg(['count',␣
↪'median'])
print(groupby_day_rev.sort_values('median'))
```

```
                     count      median
release_day_of_week
Saturday               129  41158757.0
Friday                1304  42185535.5
Sunday                 112  44367120.5
Monday                 157  49469904.0
```

```
Tuesday                223   68896829.0
Thursday               665   77000000.0
Wednesday              593   86658558.0
```

By ranking, we see budget is the most correlated with revenue, followed by popularity and vote count. Runtime is not very strongly correlated with revenue. Correlation with vote average is suprisingly low. Correlation with id is, as expected, very low.

Modeling and Question Answering

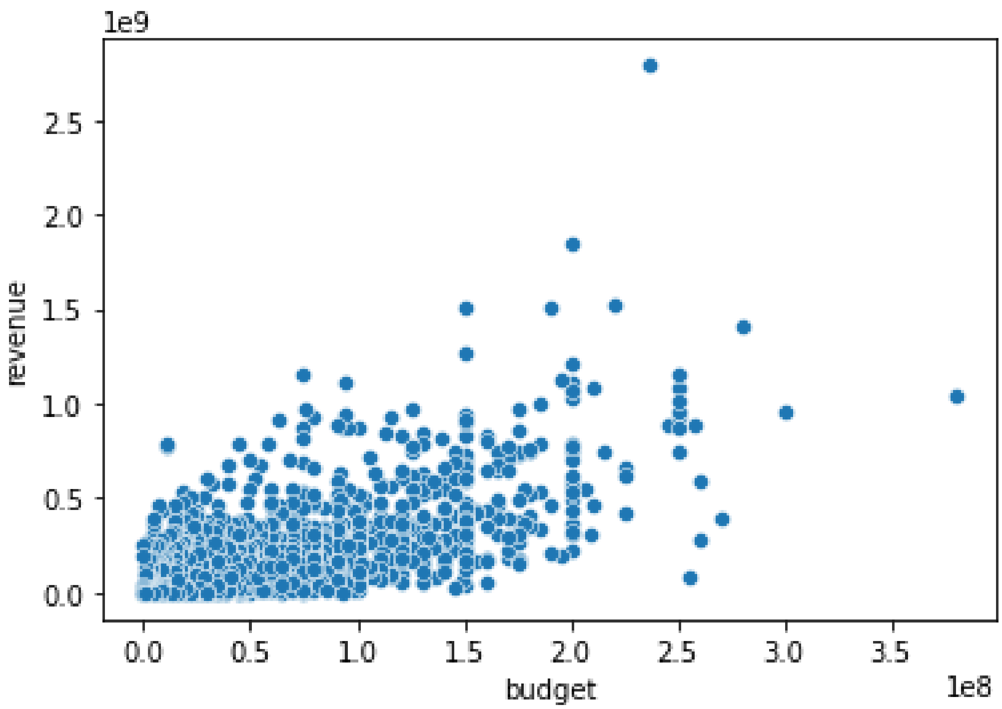
[22]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
import math
```

- Movie Revenue Prediction Model 1 – Zhe Zhou

[23]:
```
# Create a scatter plot between budget and revenue to find out outliers.
sns.scatterplot(data["budget"], data["revenue"])
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[23]:
```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```

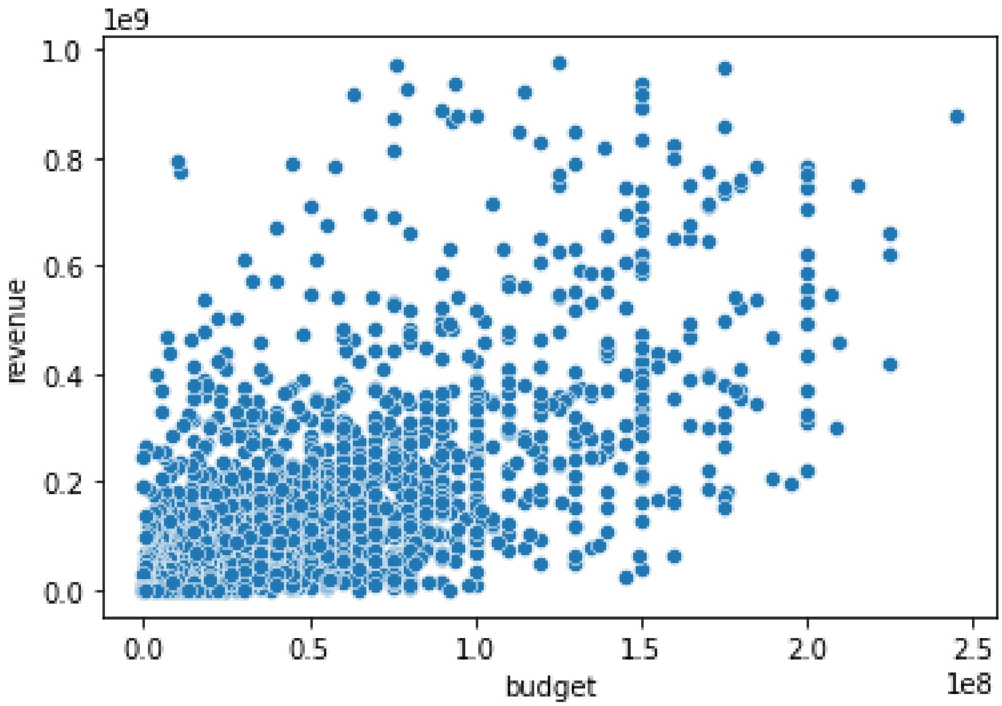


[24]:
```
# Remove the outliers according to the scatter plot.
data_without_outliers = data[(data["budget"] < 250000000) & (data["revenue"] <␣
↪1000000000)]
```

[25]:
```
# Create a scatter plot again, and check if there are any outliers else.
sns.scatterplot(data_without_outliers["budget"],␣
↪data_without_outliers["revenue"])
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[25]: `<AxesSubplot:xlabel='budget', ylabel='revenue'>`



Build model without the Cross-Validation

[26]:
```
# Create a LinearRegression obeject.
model_1 = LinearRegression()
# Create some empty lists to store values.
```

```
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    # Seperate the dataset to training set and test set.
    train, test = train_test_split(data_without_outliers)
    # Fit the dataset to the model.
    model_1 = model_1.fit(train["budget"].to_numpy().reshape(-1, 1),␣
 ↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_1.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_1.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_1.predict(test["budget"].to_numpy().reshape(-1, 1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
    times = times + 1
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
 ↪model.
model_1.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_1.intercept_ = np.mean(intercept)
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_1.coef_[0]) + " *␣
 ↪Budget + " + str(model_1.intercept_))
```

```
The final linear model is: Revenue = 2.566656386374823 * Budget +
12329307.9937768
```

[27]:
```
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE:", MSE_average)
# Calculate RMSE.
RMSE = np.sqrt(MSE_average)
# Print out the RMSE.
print("RMSE:", RMSE)
```

```
Average MSE: 1.3072347987637014e+16
RMSE: 114334369.23181504
```

Here we use standard linear regression. With some outliers removed, RMSE comes out to be 113 million USD, a 32 precent improvement from guessing median only.

The equation is roughly Revenue = 2.571 * Budget + 12 Million USD

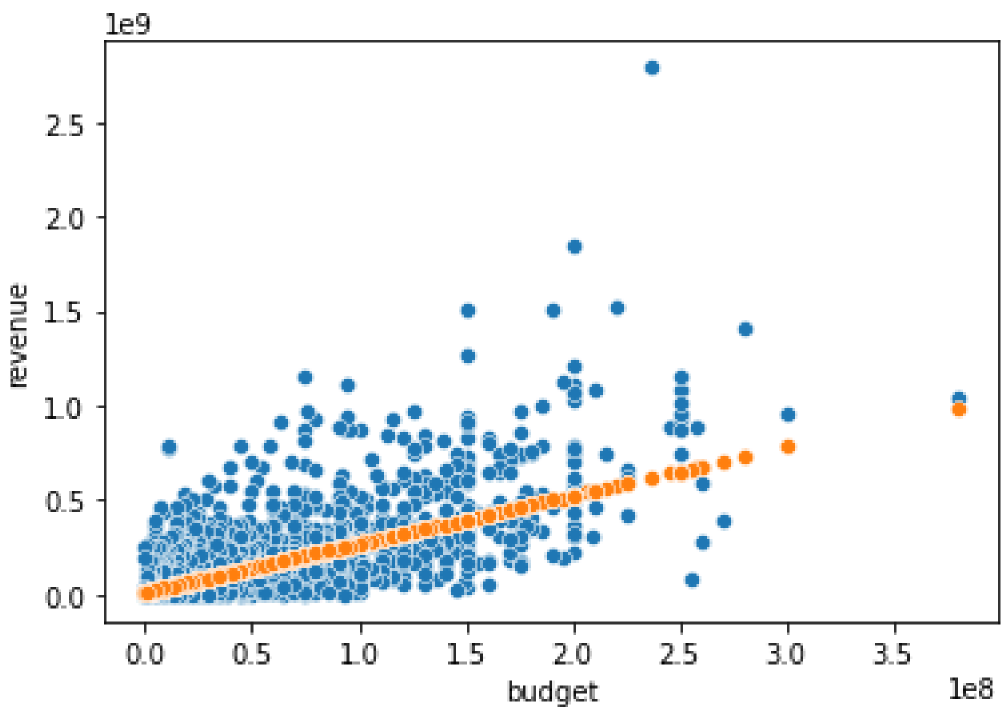
Without removing the outliers, we get a RSME 132 million.

[28]:
```python
# By the model, calculate the predicted values of revenue.
predictions = model_1.predict(data["budget"].to_numpy().reshape(-1, 1))
```

[29]:
```python
# Create a scatter plot between budget and revenue.
sns.scatterplot(data["budget"], data["revenue"])
# Create a scatter plot between budget and predictions.
sns.scatterplot(data["budget"], predictions)
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[29]: <AxesSubplot:xlabel='budget', ylabel='revenue'>



Build model with the Cross-Validation

[30]:
```
# Create a LinearRegression obeject.
model_1 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a KFold object to separate the data to the Cross-Validation set.
kf = KFold(n_splits = 10, shuffle = True)
# Create a loop to do the Cross-Validation.
for train_index, test_index in kf.split(data_without_outliers):
    # Get a train set.
    train = data_without_outliers.iloc[train_index]
    # Get a test set.
    test = data_without_outliers.iloc[test_index]
    # Fit the dataset to the model.
    model_1 = model_1.fit(train["budget"].to_numpy().reshape(-1, 1),␣
↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_1.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_1.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_1.predict(test["budget"].to_numpy().reshape(-1, 1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
↪model.
model_1.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_1.intercept_ = np.mean(intercept)
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_1.coef_[0]) + " *␣
↪Budget + " + str(model_1.intercept_))
```

```
The final linear model is: Revenue = 2.5712674158206705 * Budget +
12126093.782531265
```

[31]:
```
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE:", MSE_average)
# Calculate RMSE.
RMSE = np.sqrt(MSE_average)
# Print out the RMSE.
print("RMSE:", RMSE)
```

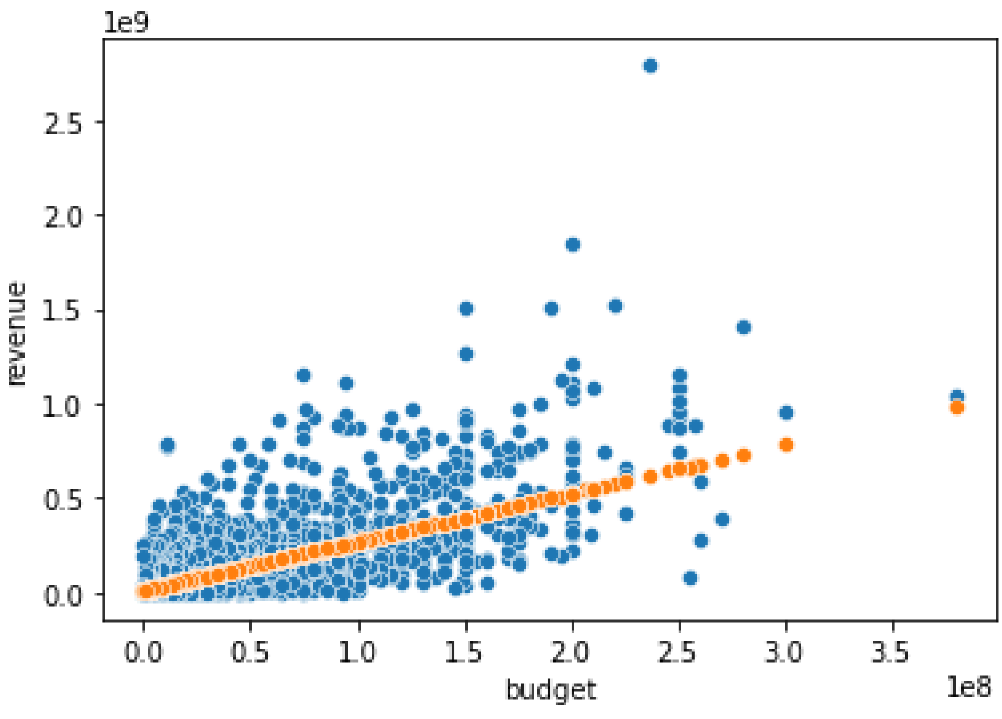
```
Average MSE: 1.2944168346875536e+16
RMSE: 113772441.06933601
```

[32]:
```python
# By the model, calculate the predicted values of revenue.
predictions = model_1.predict(data["budget"].to_numpy().reshape(-1, 1))
```

[33]:
```python
# Create a scatter plot between budget and revenue.
sns.scatterplot(data["budget"], data["revenue"])
# Create a scatter plot between budget and predictions.
sns.scatterplot(data["budget"], predictions)
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[33]: <AxesSubplot:xlabel='budget', ylabel='revenue'>



From the table and heat map, we find the correlation coefficient between budget and revenue is highest. Therefore, we want to build a linear regression model between them.

Before training the model, we have to separate the dataset into a training set and a test set, and then we need to use the fit function to generate an appropriate linear regression model. But, we find the value of MSE is unsteady and inaccurate each time we generate a linear regression. Since there existing extreme cases which will impact our linear model, we repeat the process of fitting the training set a hundred times and record the value of coefficient (slope) and intercept. And then, we use the mean of coefficient (slope) and intercept as the final model. In order to evaluate the performance of the model, we calculate MSE according to the test set and record value each time we generate the linear regression model. And then, we use the mean as Mean Square Error to the final model.

However, after evaluating the performance of this model, we find that the value of MSE is very large, so it implies that the final model's predictions are not so accurate. But, we wonder if the Cross-Validation can help increase the model's accuracy even though we have applied a similar method. After using the Cross-Validation, we calculate the MSE again. Unfortunately, the MSE doesn't have an obvious decrease.

Therefore, the final model cannot provide accurate predictions, and we think the reason is that we do not apply other features such as genres, production companies to the model. Hence, we guess these variables also play significant roles in movies' revenue.

- Movie Revenue Prediction Model 2 – Oliver Liu

Baseline Model - Guessing the mean of ''training set"

If we do this, then our avg RMSE will be, in essence, the standard deviation of the revenue, which is 187 million USD. Although out of order, we also calculated the RMSE if we guess the median, which is shown a few lines below, rather than the mean. It was using a somewhat unconventional coding style. The RMSE for median came out to be 165 million USD

Code for RMSE for predicting median

[34]:
```
# BASELINE MODEL - MEDIAN
basem1 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    MSE.append(np.mean((test["revenue"] - np.
 ↪median(data_without_outliers['revenue'])) ** 2))
    times = times + 1
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
 ↪model.
basem1.coef_ = 0
# Use the average of the value of intercept as the intercept of the fianl model.
basem1.intercept_ = np.median(data_without_outliers['revenue'])
# Print the final linear regression model.
```

```
print("predicting median every time is: Revenue = " + str(np.
 ↪median(data_without_outliers['revenue'])) )
```

```
predicting median every time is: Revenue = 55707411.0
```

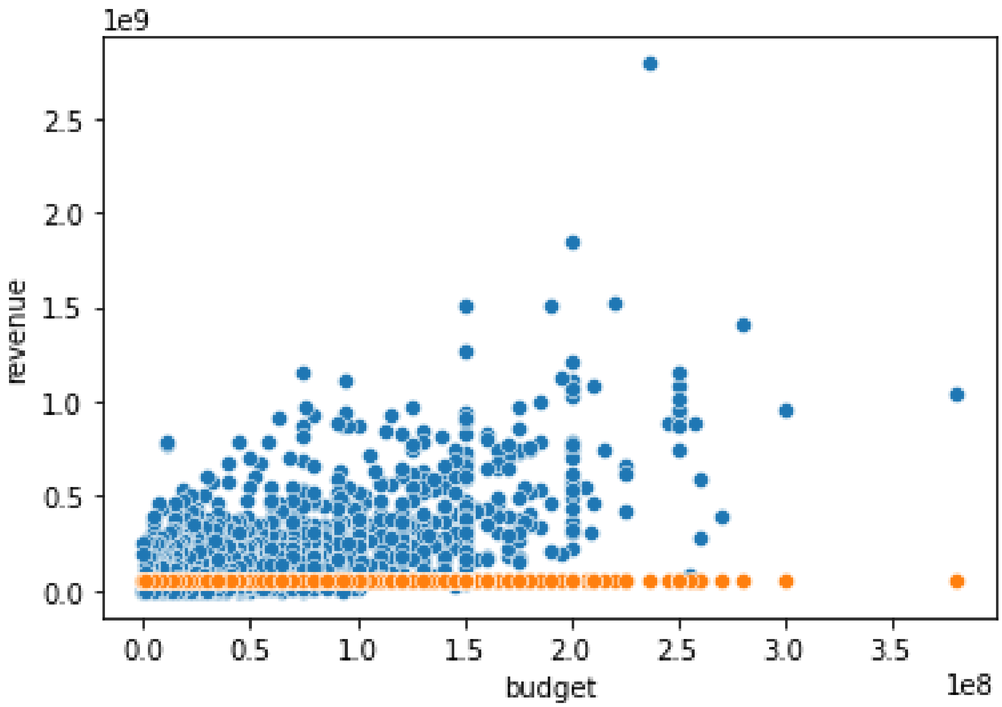
[35]:
```
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE, for basem1:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
```

```
Average MSE, for basem1: 2.7440275506563172e+16
Average RMSE: 165.65106551593072 million
```

[36]:
```
# Create a scatter plot between budget and revenue.
sns.scatterplot(data["budget"], data["revenue"])
# Create a scatter plot between budget and predictions.
sns.scatterplot(data["budget"], np.median(data_without_outliers['revenue']))
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[36]:
```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



Advanced Model - Applying a non-linear function to budget

Here we try training by applying a non-linear function to budget, to see if we can obtain a better model. For simplicity, we have called all the non-linear transformations 'budgetSquared' We used the original dataset, without removing outliers. First off, is an identity transformation, so same as the standard linear regression, except without removing any outliers. RMSE: 132.77 million USD

After trying several functions, including squared, cubed, square root… The best one came out to be raising budget to the 1.25th power. That resulted in a RMSD of 129 million USD, not a big improvement from 132 million, not enough to pass Occam's Razor's test.

[37]:
```
data["budgetSquared"] = data["budget"]
# Create a LinearRegression obeject.
model_2 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    # Seperate the dataset to training set and test set.
    train, test = train_test_split(data)
    # Fit the dataset to the model.
```

```python
    model_2 = model_2.fit(train["budgetSquared"].to_numpy().reshape(-1, 1),␣
↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_2.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_2.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_2.predict(test["budgetSquared"].to_numpy().reshape(-1,␣
↪1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
    times = times + 1
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
↪model.
model_2.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_2.intercept_ = np.mean(intercept)
# # By the model, calculate the predicted values of revenue.
predictions = model_2.predict(data["budgetSquared"].to_numpy().reshape(-1, 1))
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_2.coef_[0]) + " *␣
↪BudgetSquared + " + str(model_2.intercept_))
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
sns.scatterplot(data["budget"], data["revenue"])
sns.scatterplot(data["budget"], predictions)
```

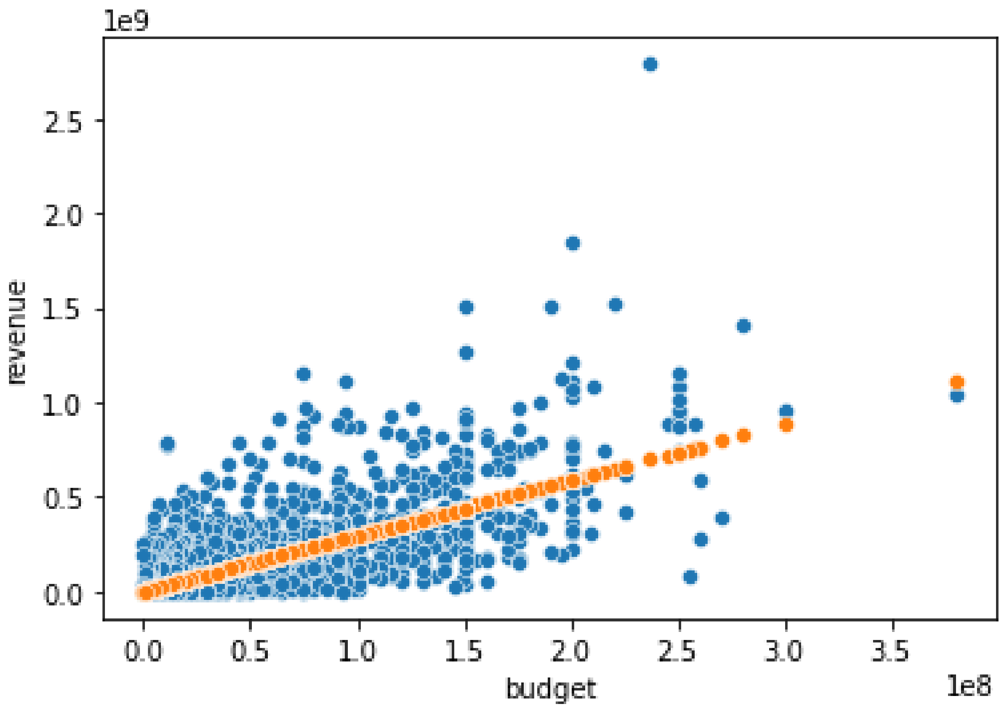
```
<ipython-input-37-025ba891d088>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data["budgetSquared"] = data["budget"]
```

```
The final linear model is: Revenue = 2.943523402423202 * BudgetSquared +
1507493.7742016285
Average MSE: 1.806593412697941e+16
Average RMSE: 134.40957602410407 million
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[37]:
```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



[38]:
```python
data["budgetSquared"] = data["budget"] ** 1.25
# Create a LinearRegression obeject.
model_2 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    # Seperate the dataset to training set and test set.
    train, test = train_test_split(data)
    # Fit the dataset to the model.
```

```
    model_2 = model_2.fit(train["budgetSquared"].to_numpy().reshape(-1, 1),␣
↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_2.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_2.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_2.predict(test["budgetSquared"].to_numpy().reshape(-1,␣
↪1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
    times = times + 1
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
↪model.
model_2.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_2.intercept_ = np.mean(intercept)
# By the model, calculate the predicted values of revenue.
predictions = model_2.predict(data["budgetSquared"].to_numpy().reshape(-1, 1))
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_2.coef_[0]) + " *␣
↪BudgetSquared + " + str(model_2.intercept_))
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
sns.scatterplot(data["budget"], data["revenue"])
sns.scatterplot(data["budget"], predictions)
```

```
<ipython-input-38-bd3cf0b8d27f>:1: 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#returning-a-view-versus-a-copy
  data["budgetSquared"] = data["budget"] ** 1.25
```

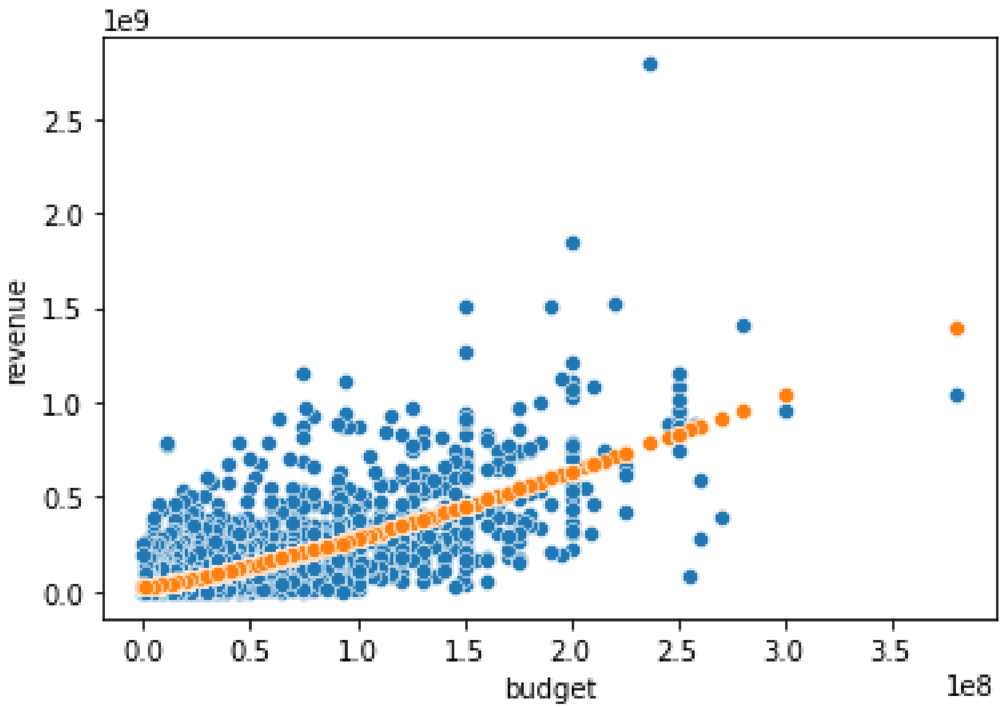
```
The final linear model is: Revenue = 0.025778384962042742 * BudgetSquared +
25218258.578622248
Average MSE: 1.74018397337291e+16
Average RMSE: 131.9160328911126 million
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[38]:

```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



Advanced model - classifying by budget, then applying linear regression

We created three budget classes, under 15 million USD, 15 million - 105 million USD, and Over 105 million USD, as data1, data2, data3, respectively. We then applied standard linear regression to those three classes, and compared it to the original model 1. As it turned out, model 1 was nearly identical to what data1 and data2 training separately, but for model 3, Model 1 RMSE: 312 million, where training on data3 alone 292 million, so that gave a 6.4 percent improvement, perhaps still not enough to pass Occam's Razor Test.

[39]:

```
data1 = data[(data["budget"] < 15000000)]
data2 = data[(data["budget"] > 15000000) & (data["budget"] < 105000000)]
data3 = data[(data["budget"] > 105000000)]
# data4 = data[(data["budget"] > 200000000)]
# data1.describe()
#data2.describe()
```

```python
# data3.describe()
# data4.describe()
# pyplt.hist(data1.budget, bins=10)
# pyplt.hist(data2.budget, bins=10)
# pyplt.hist(data3.budget, bins=10)
# pyplt.hist(data4.budget, bins=10)
```

[40]:

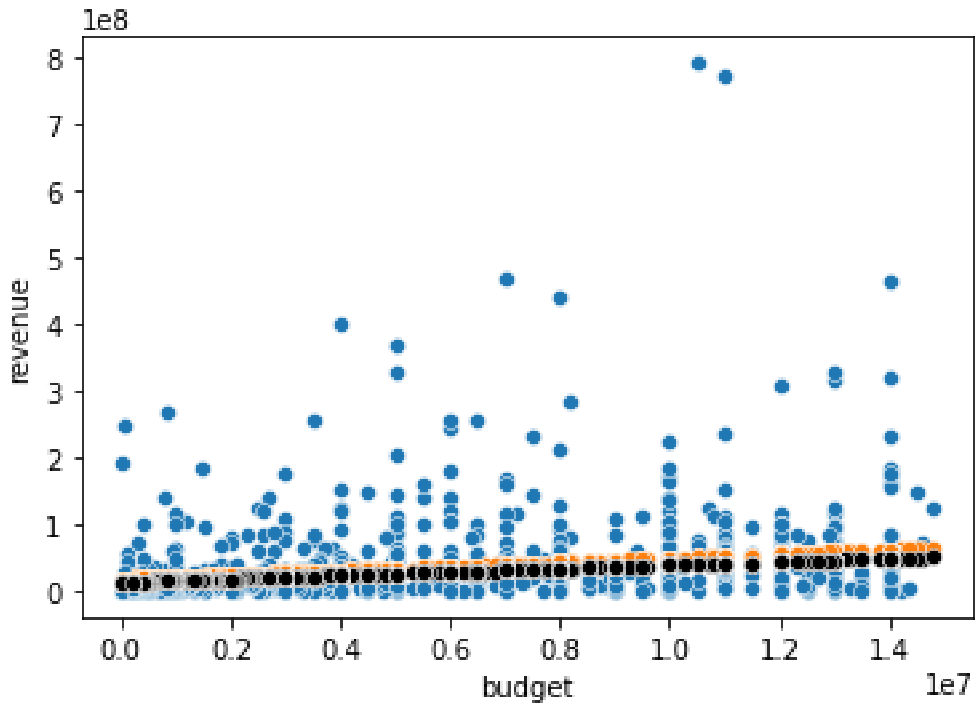
```python
# Create a LinearRegression obeject.
model_31 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    # Seperate the dataset to training set and test set.
    train, test = train_test_split(data1)
    # Fit the dataset to the model.
    model_31 = model_31.fit(train["budget"].to_numpy().reshape(-1, 1),␣
 ↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_31.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_31.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_31.predict(test["budget"].to_numpy().reshape(-1, 1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
    times = times + 1
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
 ↪model.
model_31.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_31.intercept_ = np.mean(intercept)
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_31.coef_[0]) + " *␣
 ↪Budget + " + str(model_31.intercept_))
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
data1["predictions"] = model_31.predict(data1["budget"].to_numpy().reshape(-1,␣
 ↪1))
print("Average MSE:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
model1mse = (data1['revenue'] - (data1["budget"] * 2.5666456407290505 +␣
 ↪12245913.01169521)) ** 2
```

```python
print("Model 1 RMSE:", np.mean(model1mse) ** 0.5 / 1000000, 'million')
sns.scatterplot(data1["budget"], data1["revenue"])
sns.scatterplot(data1["budget"], data1["predictions"])
sns.scatterplot(data1["budget"], data1["budget"] * 2.5666456407290505 +␣
↪12245913.01169521, color='Black')
```

```
The final linear model is: Revenue = 3.17008872400231 * Budget +
16941234.099459354
Average MSE: 3886018272397437.0
Average RMSE: 62.33793606141799 million
Model 1 RMSE: 64.5022647018798 million
```

```
<ipython-input-40-c0e7cc01a8cf>:32: 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#returning-a-view-versus-a-copy
  data1["predictions"] = model_31.predict(data1["budget"].to_numpy().reshape(-1,
1))
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[40]:
```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



[41]:
```python
# Create a LinearRegression obeject.
model_32 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    # Seperate the dataset to training set and test set.
    train, test = train_test_split(data2)
    # Fit the dataset to the model.
    model_32 = model_32.fit(train["budget"].to_numpy().reshape(-1, 1),␣
 ↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_32.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_32.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_32.predict(test["budget"].to_numpy().reshape(-1, 1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
    times = times + 1
```

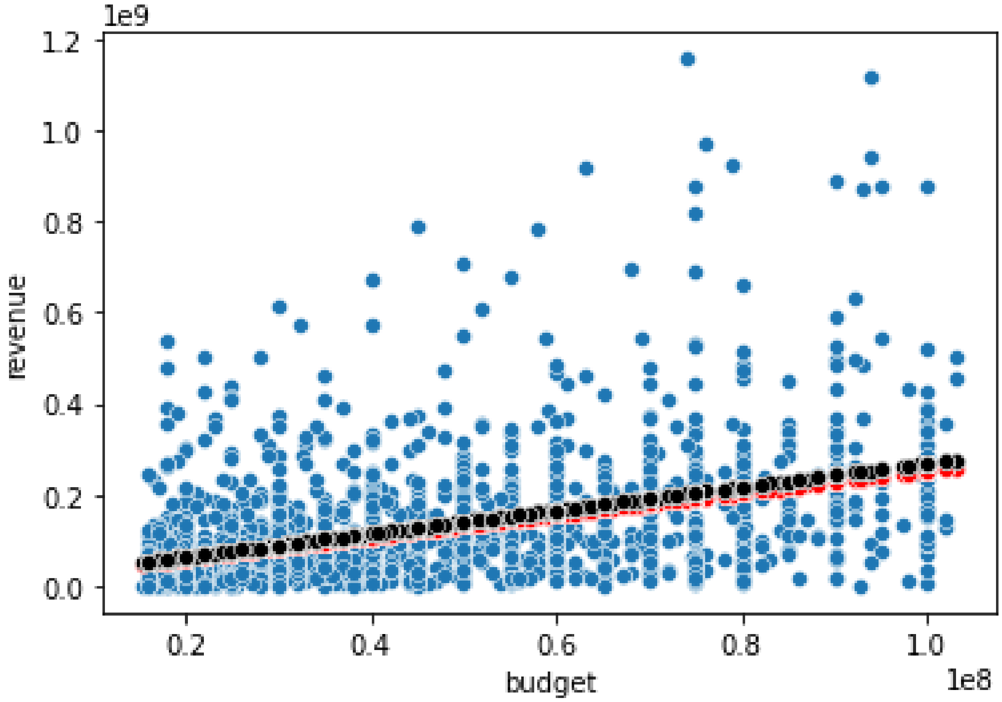
```python
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
↪model.
model_32.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_32.intercept_ = np.mean(intercept)
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_32.coef_[0]) + " *␣
↪Budget + " + str(model_32.intercept_))
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
data2["predictions"] = model_32.predict(data2["budget"].to_numpy().reshape(-1,␣
↪1))
print("Average MSE:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
model1mse = (data2['revenue'] - (data2["budget"] * 2.5666456407290505 +␣
↪12245913.01169521)) ** 2
print("Model 1 RMSE:", np.mean(model1mse) ** 0.5 / 1000000, 'million')
sns.scatterplot(data2["budget"], data2["revenue"])
sns.scatterplot(data2["budget"], data2["predictions"], color='Red')
sns.scatterplot(data2["budget"], data2["budget"] * 2.5666456407290505 +␣
↪12245913.01169521, color='Black')
```

```
The final linear model is: Revenue = 2.3938762728940497 * Budget +
11984910.061878866
Average MSE: 1.447981955105099e+16
Average RMSE: 120.33212185884112 million
Model 1 RMSE: 119.67720459015753 million
```

```
<ipython-input-41-da807a79bab6>:32: 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#returning-a-view-versus-a-copy
  data2["predictions"] = model_32.predict(data2["budget"].to_numpy().reshape(-1,
1))
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[41]:
```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



[42]:
```
# Create a LinearRegression obeject.
model_33 = LinearRegression()
# Create some empty lists to store values.
coef = []
intercept = []
MSE = []
# Create a loop.
times = 0
while (times <= 100):
    # Seperate the dataset to training set and test set.
    train, test = train_test_split(data3)
    # Fit the dataset to the model.
    model_33 = model_33.fit(train["budget"].to_numpy().reshape(-1, 1),␣
 ↪train["revenue"].to_numpy())
```

```python
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_33.coef_[0])
    # Store the value of intercept in the model of each loop.
    intercept.append(model_33.intercept_)
    # Calculate predicted values by the values of budget for each movie.
    predictions = model_33.predict(test["budget"].to_numpy().reshape(-1, 1))
    # Store the value of mean square value in the model of each loop.
    MSE.append(np.mean((test["revenue"] - predictions) ** 2))
    times = times + 1
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
↪model.
model_33.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_33.intercept_ = np.mean(intercept)
# Print the final linear regression model.
print("The final linear model is: Revenue = " + str(model_33.coef_[0]) + " *␣
↪Budget + " + str(model_33.intercept_))
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
data3["predictions"] = model_33.predict(data3["budget"].to_numpy().reshape(-1,␣
↪1))
print("Average MSE:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
model1mse = (data3['revenue'] - (data3["budget"] * 2.5666456407290505 +␣
↪12245913.01169521)) ** 2
print("Model 1 RMSE:", np.mean(model1mse) ** 0.5 / 1000000, 'million')
sns.scatterplot(data3["budget"], data3["revenue"])
sns.scatterplot(data3["budget"], data3["predictions"], color = 'Red')
sns.scatterplot(data3["budget"], data3["budget"] * 2.5666456407290505 +␣
↪12245913.01169521, color='Black')
# The final linear model is: Revenue = 2.5666456407290505 * Budget + 12245913.
↪01169521
```

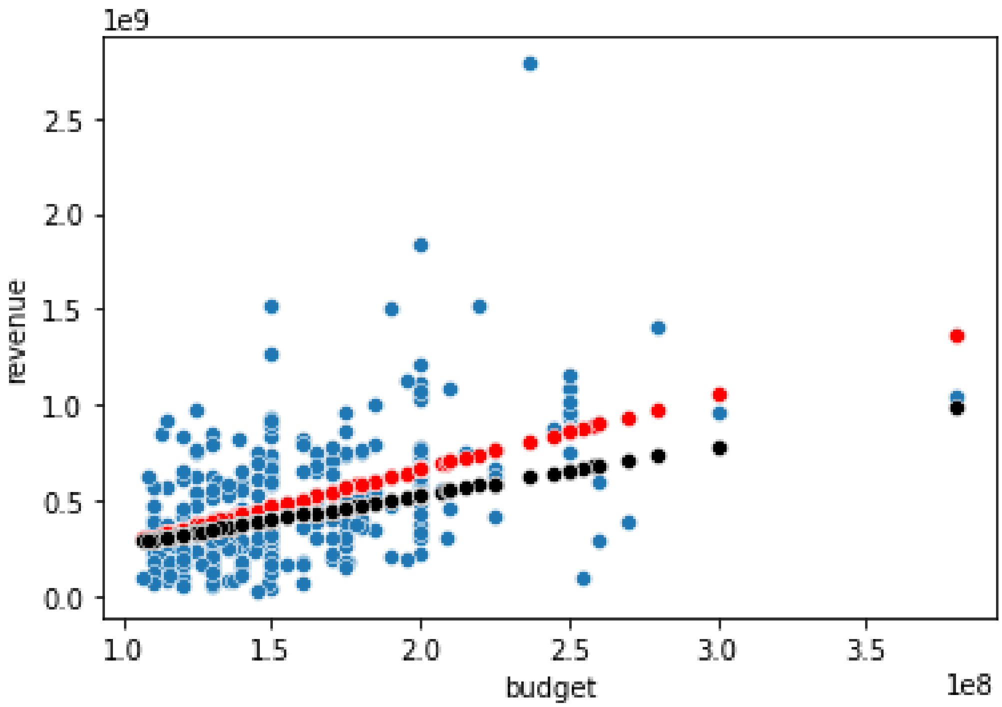
```
The final linear model is: Revenue = 3.9334203034239374 * Budget +
-123469632.8447768
Average MSE: 8.629012880410176e+16
Average RMSE: 293.75181498009806 million
Model 1 RMSE: 312.82331489501627 million
```

```
<ipython-input-42-94de11f952c7>:32: 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#returning-a-view-versus-a-copy
  data3["predictions"] = model_33.predict(data3["budget"].to_numpy().reshape(-1,
1))
```

```
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
/Users/zhezhou/opt/anaconda3/lib/python3.8/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

[42]:

```
<AxesSubplot:xlabel='budget', ylabel='revenue'>
```



- Movie Revenue Prediction Model 3 – Jeffrey Zhang

Groupy by Genre prediction of Revenue

[43]:
```
allX = {}
allY = {}
for ug in unique_genre:
    x = []
    y = []
    for l in range (0, len(data["genres"])):
        if (data["genres"].get(l) and data["revenue"].get(l) and ug in␣
 ↪data["genres"].get(l)):
            x.append(data["budget"].get(l))
            y.append(data["revenue"].get(l))
    allX[ug] = x
    allY[ug] = y
```

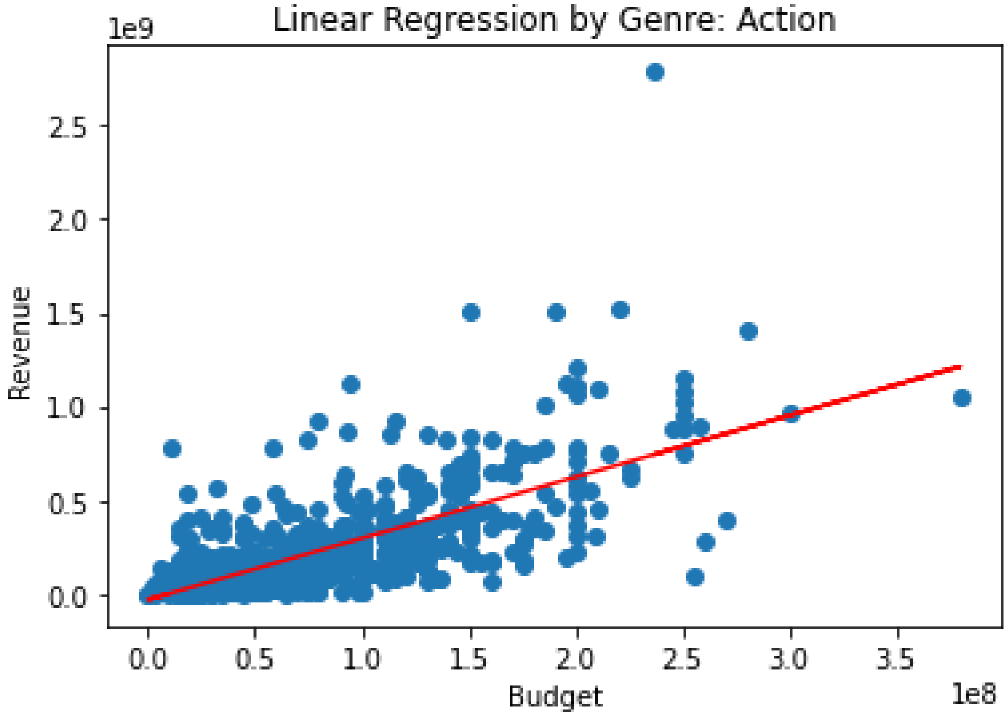
Here we sort all values of budgets (X) and revenue (Y) into two dictionaries with key values of the specific genre and the values of budget or revenue as lists depending on dictionary. In this process, repeats of the same movie does occur since most movies have more than a single genre. Additionally, using these lists would better allow us to plot the scatter plot in the future. In hindsight after the first run, the points and the line's visualization did not give a good understanding of approximation, hence the application of Napier Logarithms permitted a better visualization of data. This is why the log budget and revenue is used.

[44]:
```
for g in unique_genre:
    if allX.get(g) == []:
        allX.pop(g)
    if allY.get(g) == []:
        allY.pop(g)
```

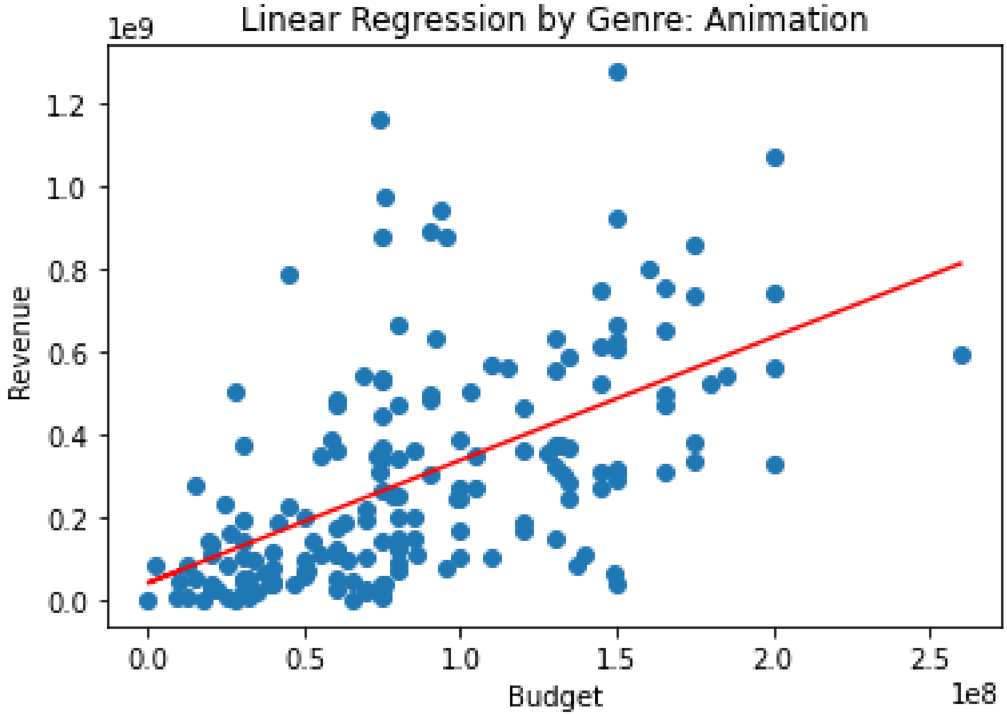
Here, we make sure that all genres have budgets and revenues by discarding the entire genre since an empty list of budget and revenues would result in an empty scatter plot.

[45]:
```
coef = []
intercept = []
# MSE = []
sum_MSE = 0
sum_RMSE = 0

for g in unique_genre:
#     MSE = []
    if(allX.get(g) and allY.get(g)):
        # Gets all coordinates for x and y
        x = np.array(allX.get(g)).reshape((-1, 1))
        y = np.array(allY.get(g))
        # Creates LinearRegression object
        m4 = LinearRegression()
        m4.fit(x,y)
        # Creates regression line based off coordinates
        y_pred = m4.predict(x)
        pyplt.scatter(x,y)
```

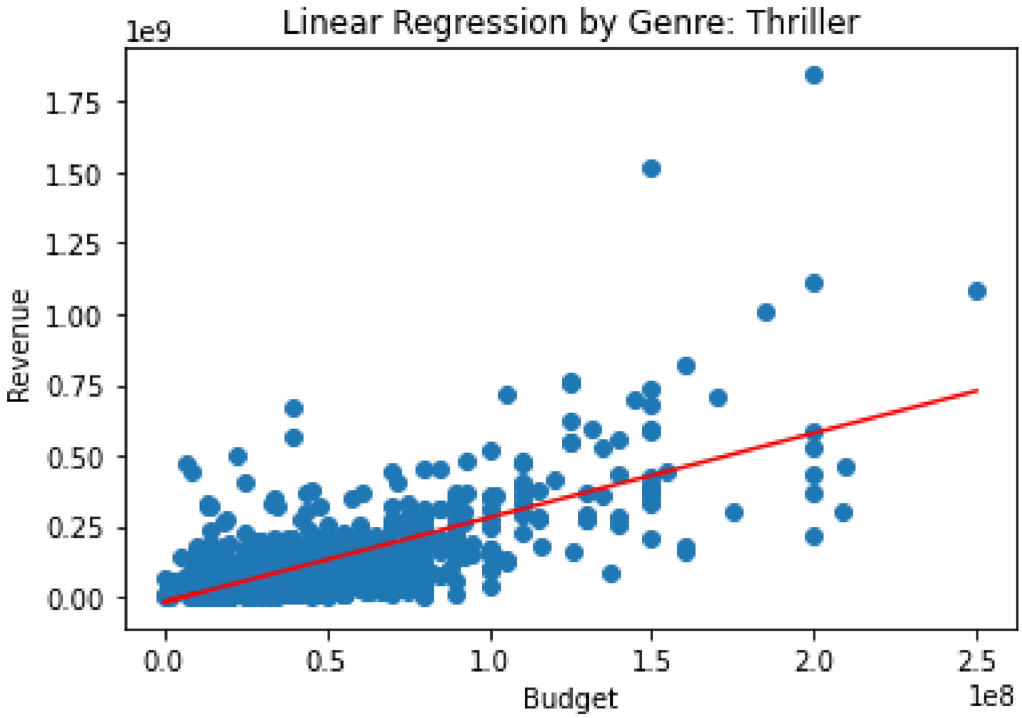
```
        # Labels and organization
        pyplt.plot(x, y_pred, color="red")
        pyplt.xlabel("Budget")
        pyplt.ylabel("Revenue")
        title = "Linear Regression by Genre: " + g
        pyplt.title(title)
        pyplt.show()

        # Gets variables for solving RMSE and MSE
        m4 = m4.fit(x,y)
        #coef.append(m4.coef_[0])
        #intercept.append(m4.intercept_)
        predictions = m4.predict(np.array(allX.get(g)).reshape(-1, 1))


        MSE = np.mean((allY.get(g) - predictions) ** 2)

        m4coef = m4.coef_[0]
        m4intercept = m4.intercept_

        #m4coef_ = np.array([np.mean(coef)])
        #m4intercept_ = np.mean(intercept)
        print("The " + g + " movie linear regression model is: Revenue = " +␣
 ↪str(m4.coef_[0]) + " * Budget + " + str(m4.intercept_))

#     MSE_average = np.mean(MSE)
        print("Average MSE, for model #4:", MSE)
        print("Average RMSE:", MSE ** 0.5 / 1000000, 'million\n')


# print("Average MSE across all genres is " +str(sum_MSE/len(allX)))
# print("Average RMSE across all genres is " +str(sum_RMSE/len(allX)) + "␣
 ↪million")
```



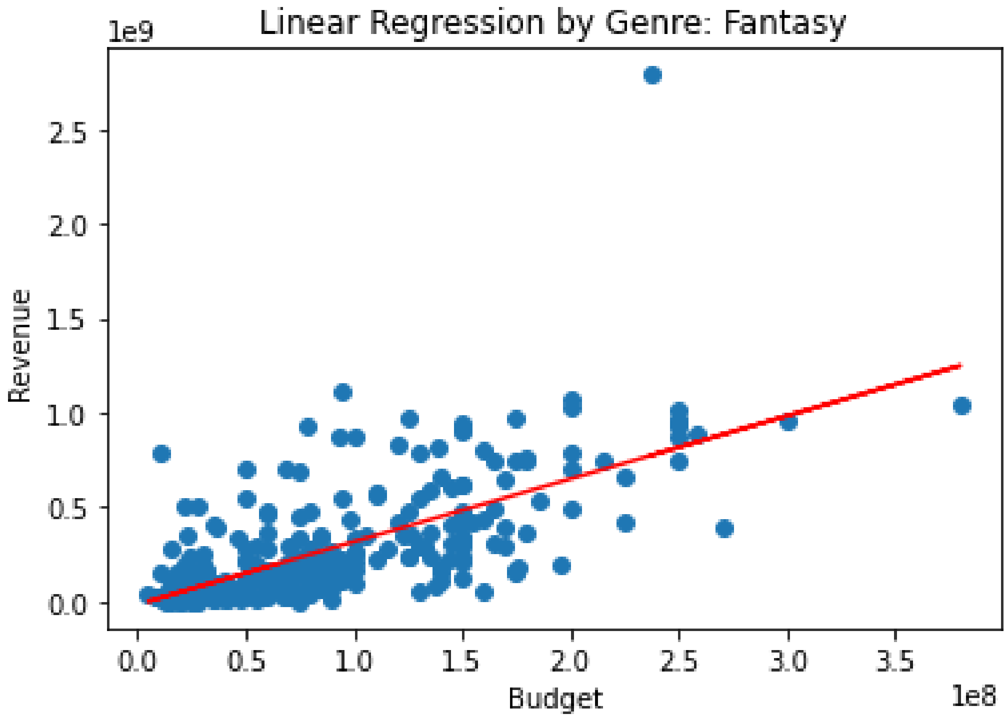

```
The Action movie linear regression model is: Revenue = 3.27536105840265 * Budget
+ -31310150.26743996
Average MSE, for model #4: 3.1220943003118476e+16
Average RMSE: 176.69449058507308 million
```



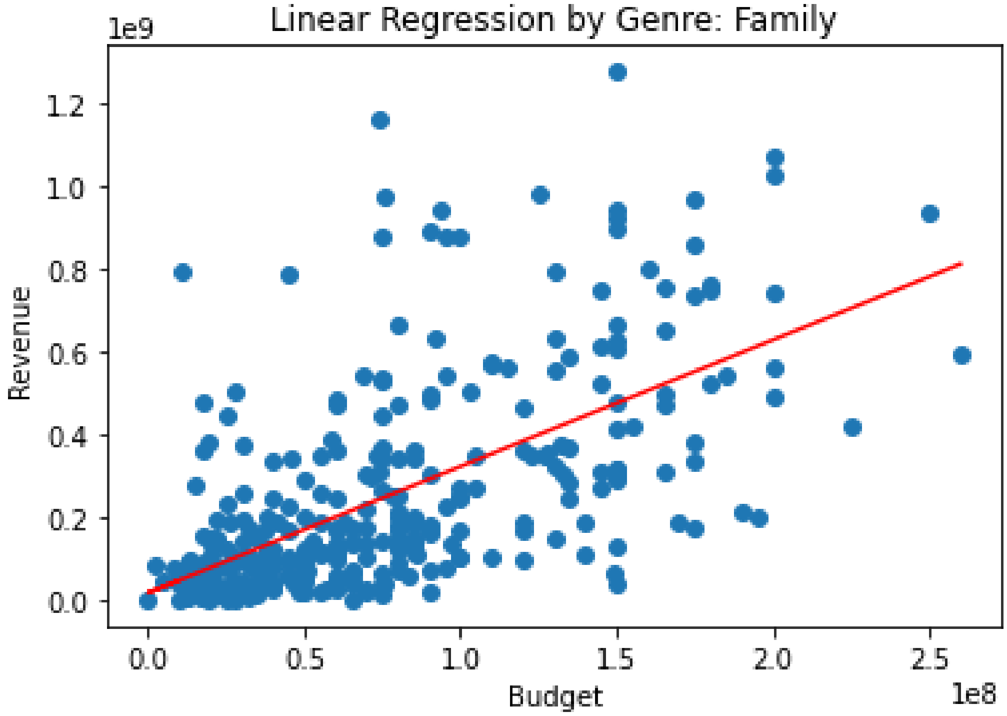
The Animation movie linear regression model is: Revenue = 2.9637210855775966 * Budget + 40371668.570864916
Average MSE, for model #4: 4.897357280337886e+16
Average RMSE: 221.2997352085602 million



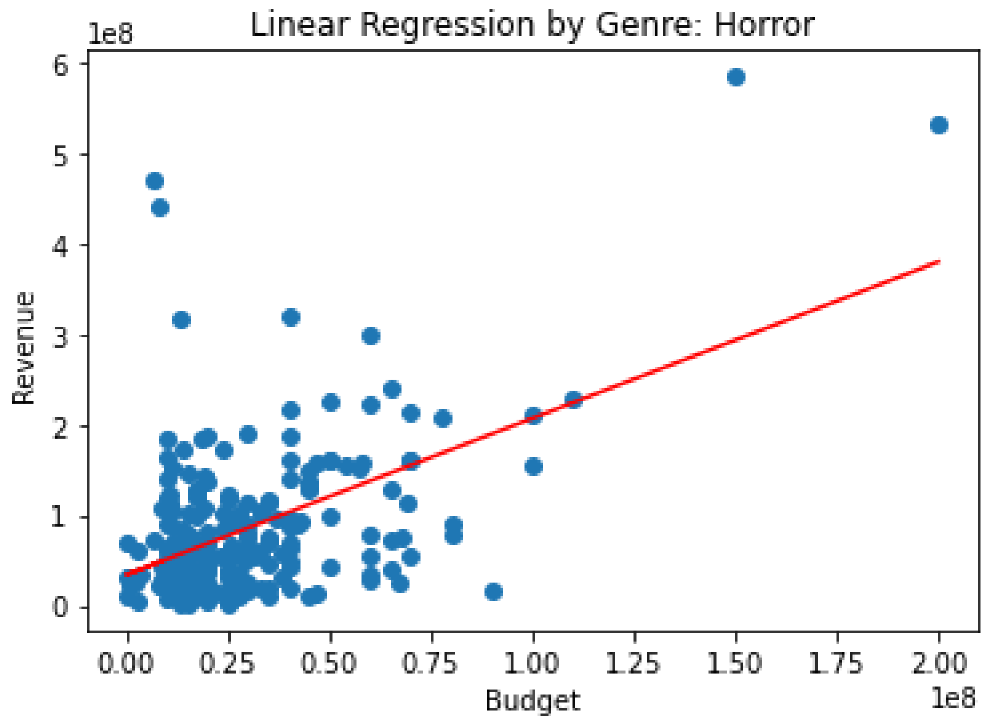

```
The Thriller movie linear regression model is: Revenue = 2.977197255722411 *
Budget + -16528402.799729079
Average MSE, for model #4: 1.4798319095074316e+16
Average RMSE: 121.64834193310782 million
```



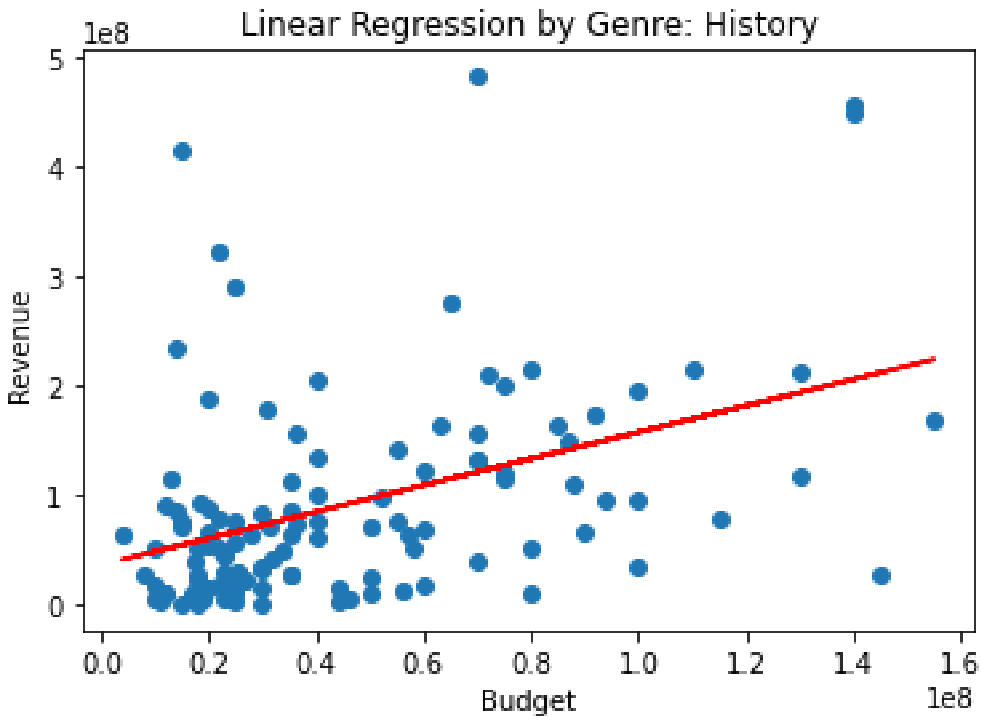

```
The Fantasy movie linear regression model is: Revenue = 3.325699778624452 *
Budget + -15003767.638961256
Average MSE, for model #4: 4.831132921485543e+16
Average RMSE: 219.79838310336913 million
```



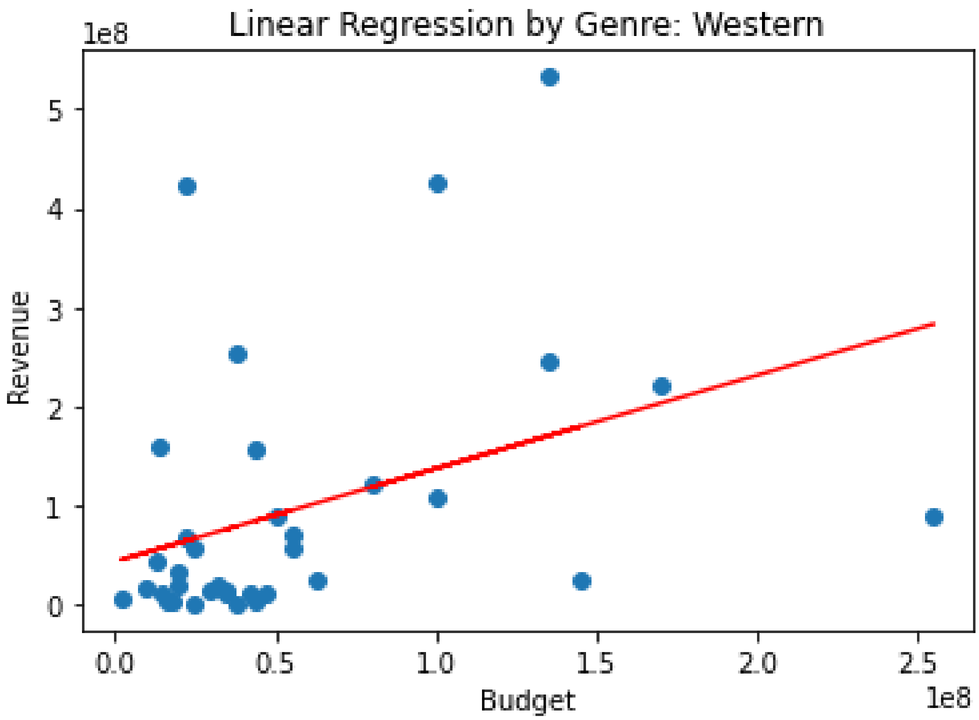

```
The Family movie linear regression model is: Revenue = 3.051889630632334 *
Budget + 16321172.228776276
Average MSE, for model #4: 3.898953910810546e+16
Average RMSE: 197.45768941245478 million
```



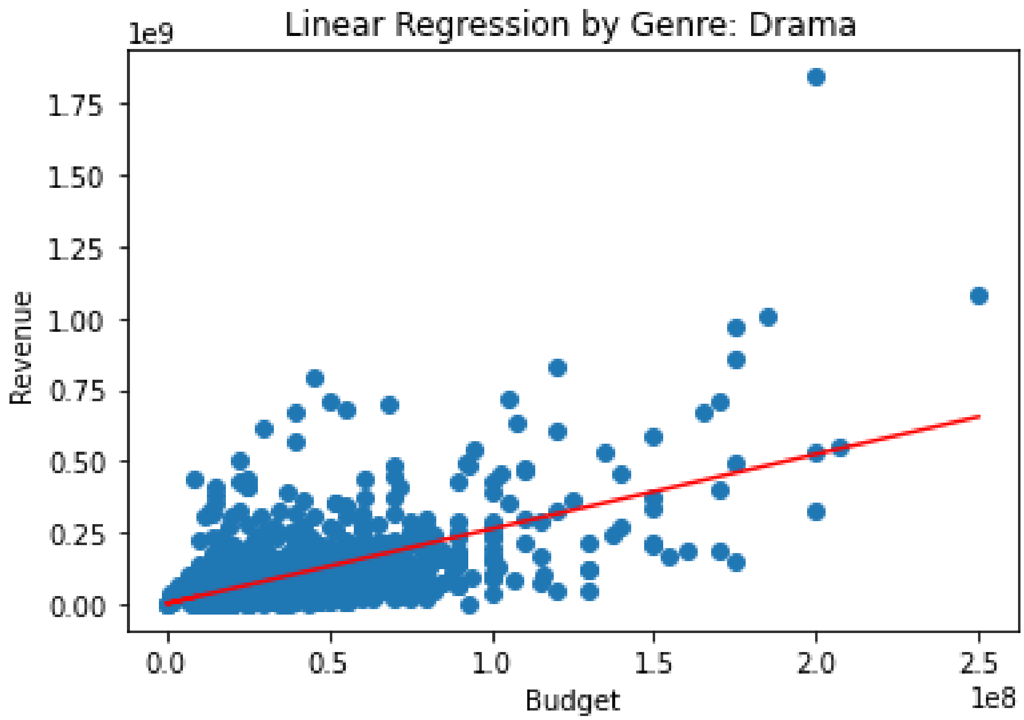

```
The Horror movie linear regression model is: Revenue = 1.7324344221729013 *
Budget + 33801689.764233895
Average MSE, for model #4: 5244335623126510.0
Average RMSE: 72.41778526803004 million
```



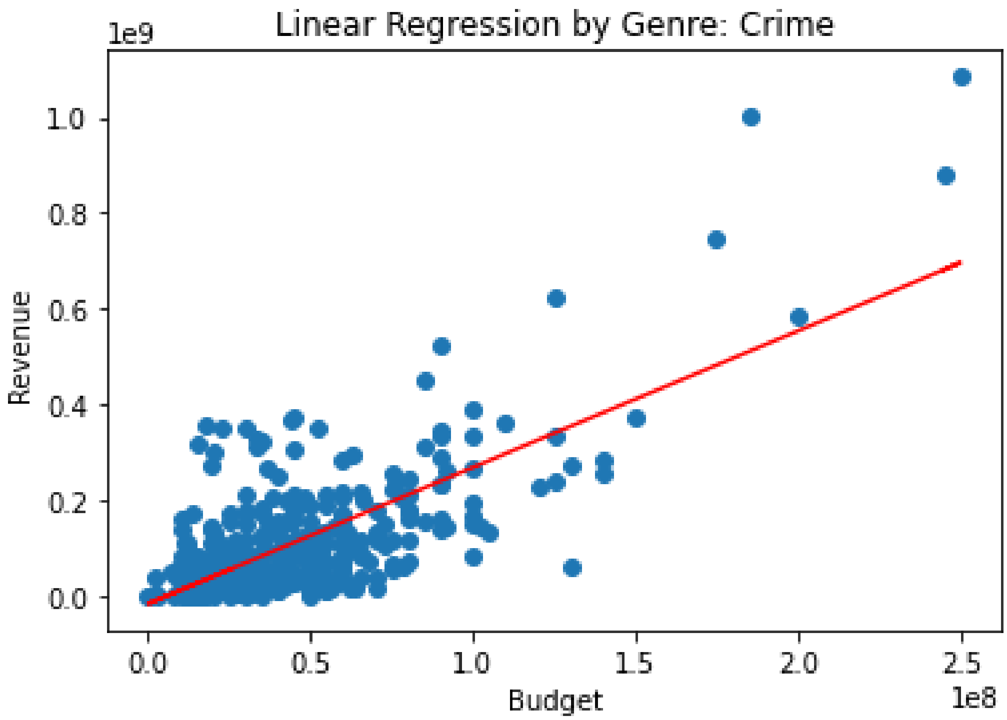
The History movie linear regression model is: Revenue = 1.2112642608985058 * Budget + 35783377.87331638
Average MSE, for model #4: 7664902602068216.0
Average RMSE: 87.54942947882766 million



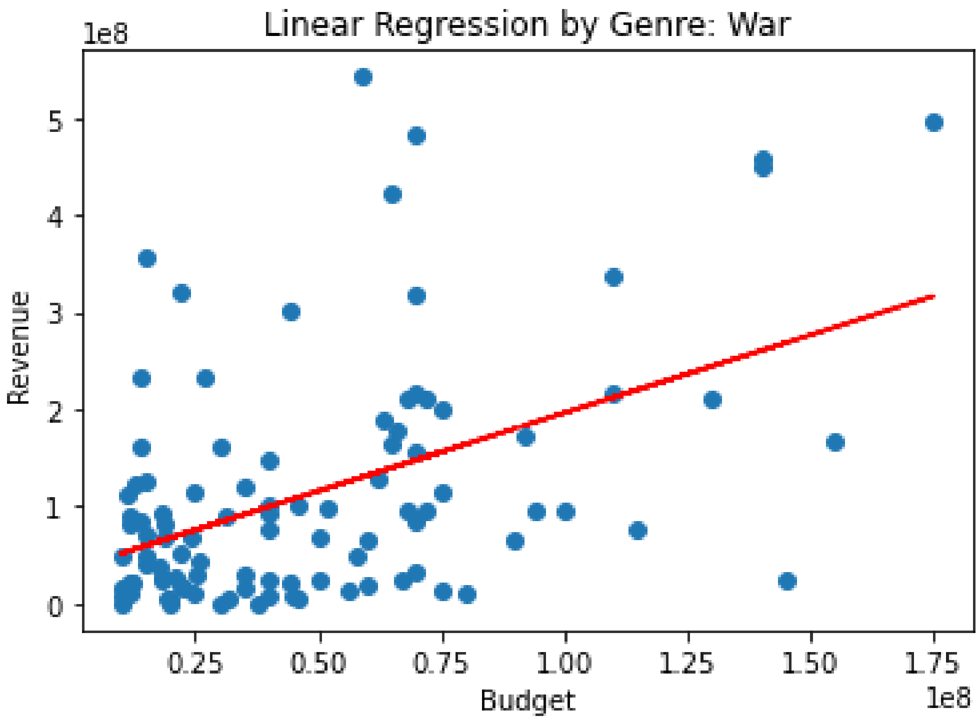
The Western movie linear regression model is: Revenue = 0.9397390871996222 * Budget + 43326583.91316391
Average MSE, for model #4: 1.494039746016042e+16
Average RMSE: 122.23091859329382 million



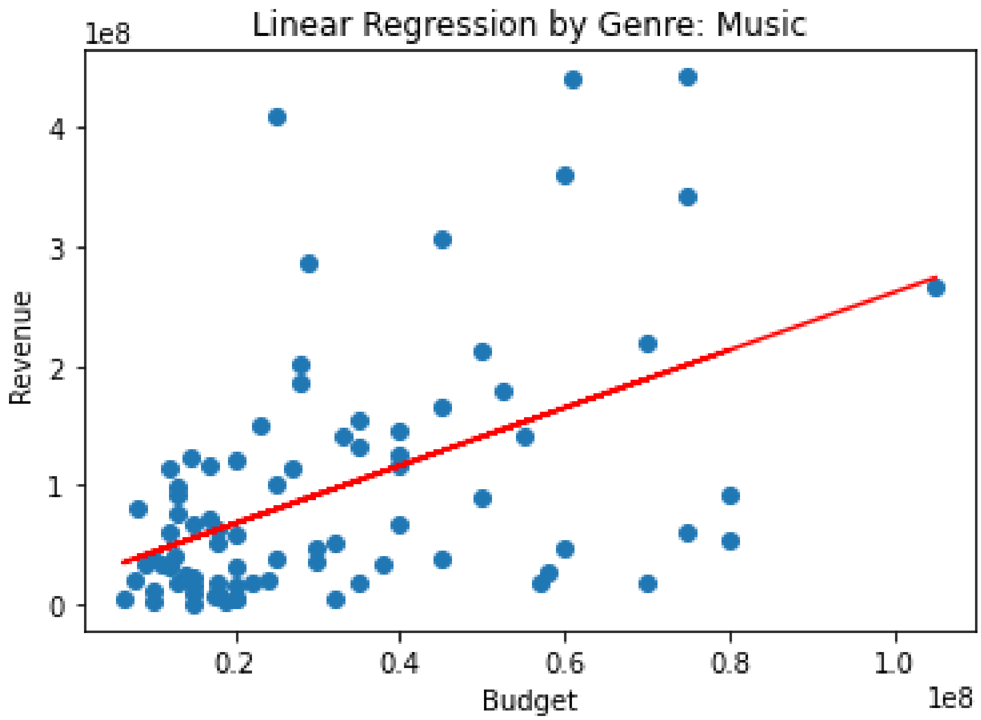

```
The Drama movie linear regression model is: Revenue = 2.6145777185957093 *
Budget + 438777.5260221511
Average MSE, for model #4: 1.2697269594628464e+16
Average RMSE: 112.68216182976107 million
```



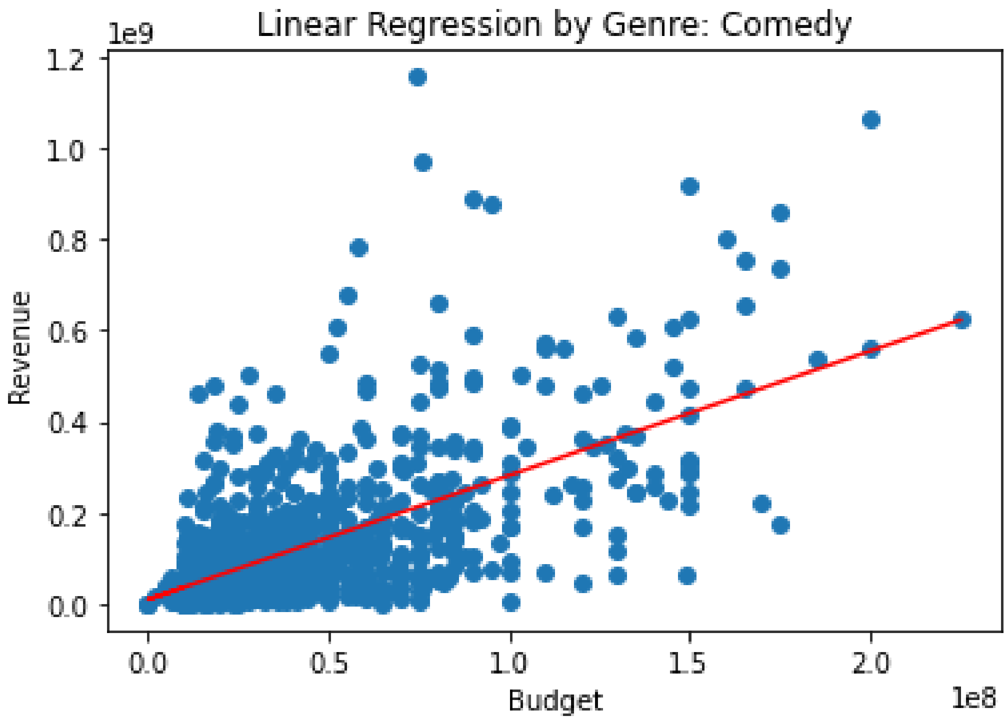

```
The Crime movie linear regression model is: Revenue = 2.8465911581112238 *
Budget + -15478316.276330054
Average MSE, for model #4: 7781101626895374.0
Average RMSE: 88.21055280914736 million
```



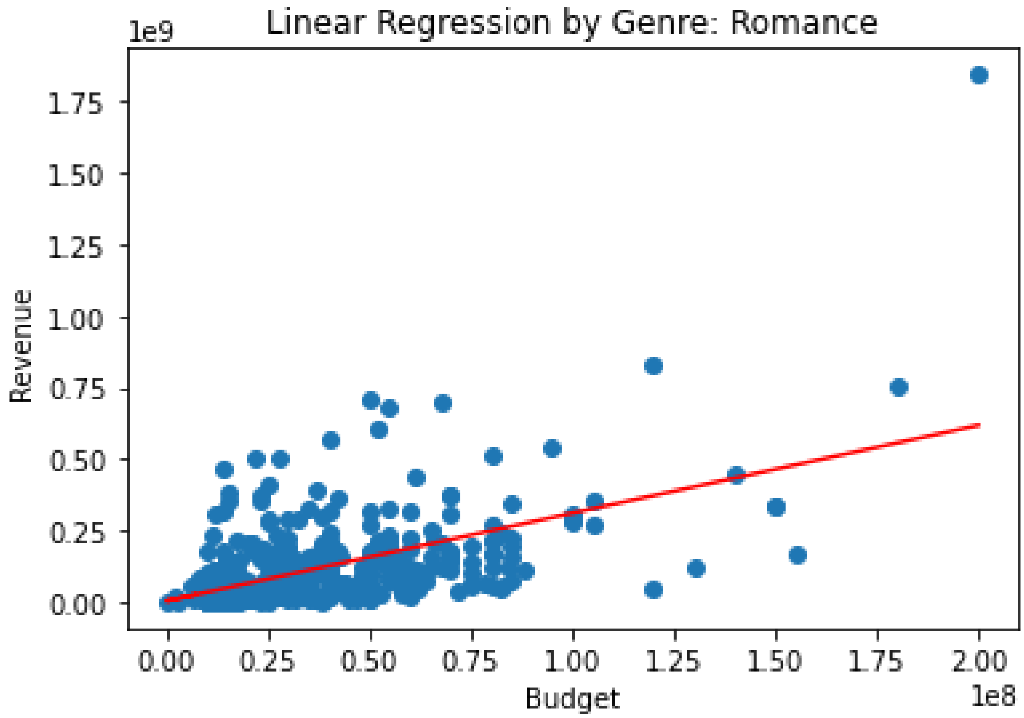

```
The War movie linear regression model is: Revenue = 1.6037701051016897 * Budget
+ 35869395.74083336
Average MSE, for model #4: 1.1855076288964162e+16
Average RMSE: 108.8810189563092 million
```



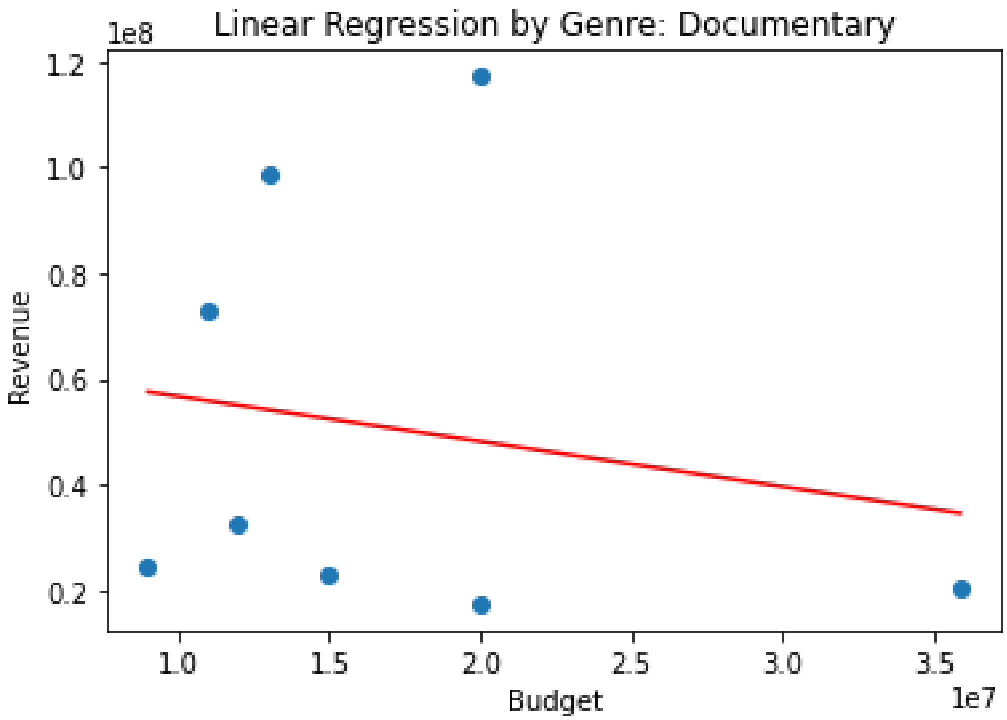
The Music movie linear regression model is: Revenue = 2.4313715680249373 * Budget + 18750612.034071982
Average MSE, for model #4: 8205279475706771.0
Average RMSE: 90.5829977187042 million



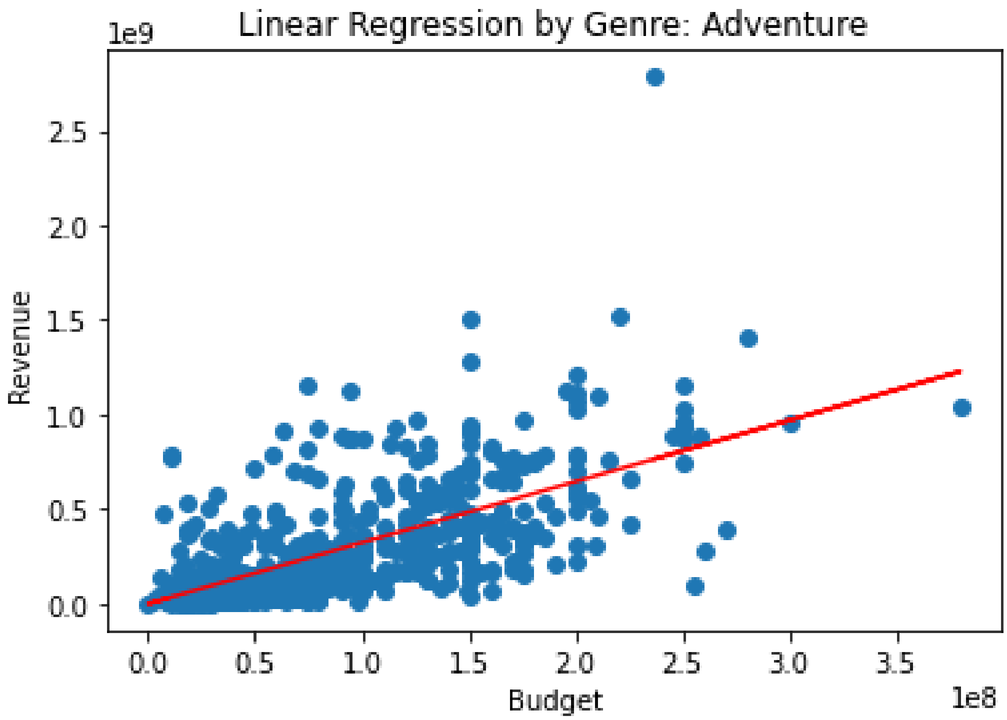

```
The Comedy movie linear regression model is: Revenue = 2.7259915650357893 *
Budget + 9181456.61353463
Average MSE, for model #4: 1.4304717514461406e+16
Average RMSE: 119.60233072336595 million
```



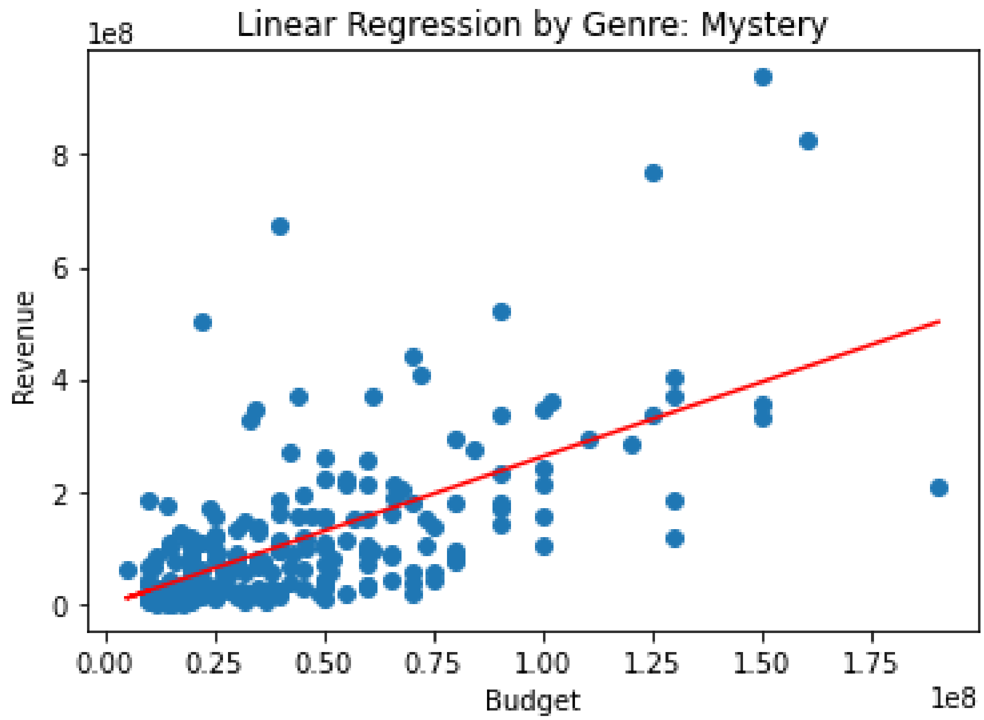

```
The Romance movie linear regression model is: Revenue = 3.07300842688342 *
Budget + 2754598.323889613
Average MSE, for model #4: 1.6090631050874286e+16
Average RMSE: 126.84885120045149 million
```



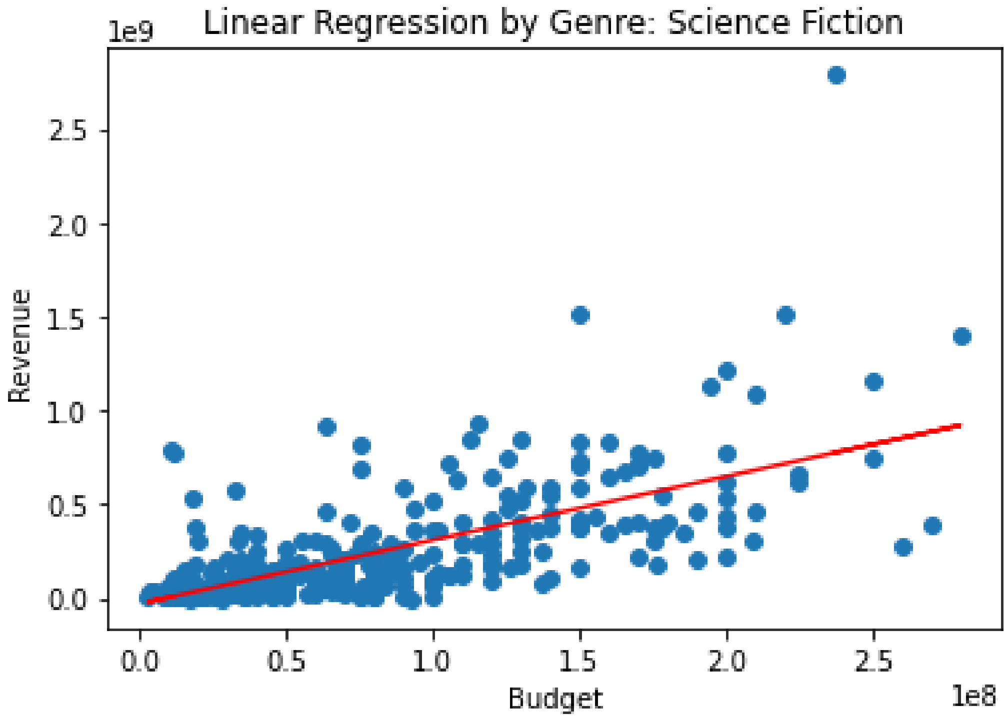

```
The Documentary movie linear regression model is: Revenue = -0.8549312637372865
* Budget + 65281408.435830235
Average MSE, for model #4: 1329788912897871.0
Average RMSE: 36.46627089377074 million
```




```
The Adventure movie linear regression model is: Revenue = 3.235762806248273 *
Budget + -2513907.053050548
Average MSE, for model #4: 4.962660402314266e+16
Average RMSE: 222.7702943014231 million
```




```
The Mystery movie linear regression model is: Revenue = 2.658592289211572 *
Budget + -3482228.860119745
Average MSE, for model #4: 1.1896080857103292e+16
Average RMSE: 109.06915630508604 million
```




```
The Science Fiction movie linear regression model is: Revenue =
3.385073525455878 * Budget + -27896196.700176716
Average MSE, for model #4: 4.547565308684023e+16
Average RMSE: 213.25021239576816 million
```

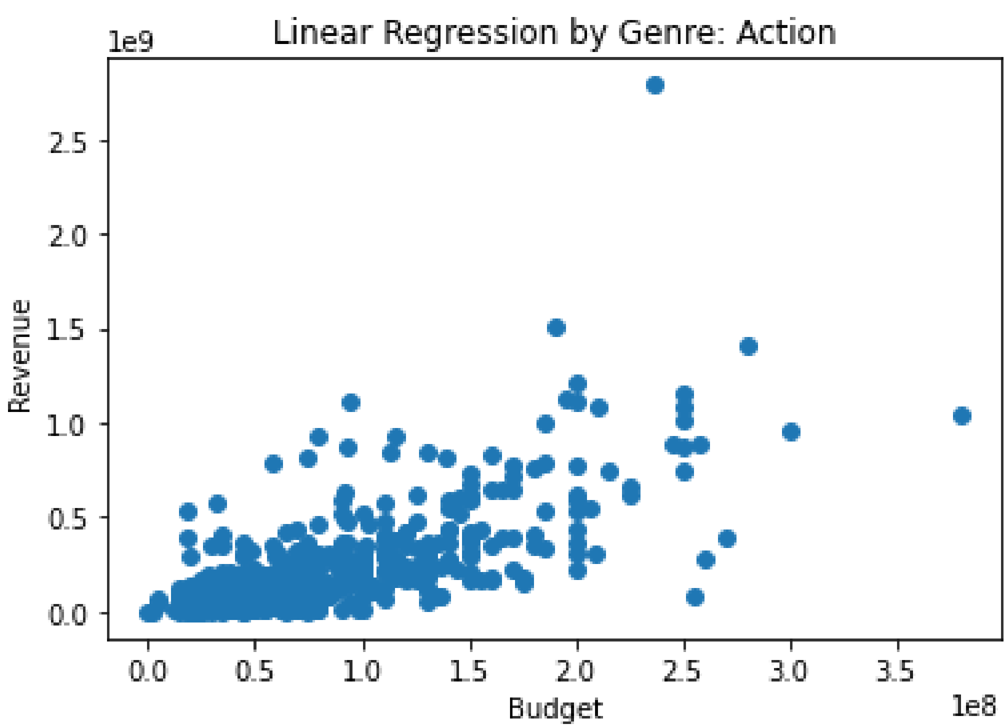
Since we have so many genres, we loop through all key values of both dictionaries to plot the scatterplot as well as labeling both axises and the name of the plot. Additionally, we use the loop to calculate the MSE as well as the RMSE to visualize the accuracy of the models. As demonstrated by the above data, all MSE are large numbers many of whom are to 10 to the 16th power. Similarly RMSE also is a large number which is no surprise since the RMSE is derivived from the MSE. With this information and the information from other models, we can determine that the MSE and RMSE are medicore to poor. To improve on this, one method can be to remove all outliers so the distance between predicted and actual values are less drastic, in other words resulting in smaller MSE and RMSE. Moreover, with this result, this is an example of a linear regression algorithmm which sorts the data into categories to achieve a better idea of what the revenue would be like per genre since each genre's basis of revenue and budget is different. This is much like how spliting the demographic of an out of school income by major would give a better idea of how much a student is suppose to earn compared to an average for the entire school.

[46]:
```
coef = []
intercept = []
MSE = []
sum_MSE = 0
```

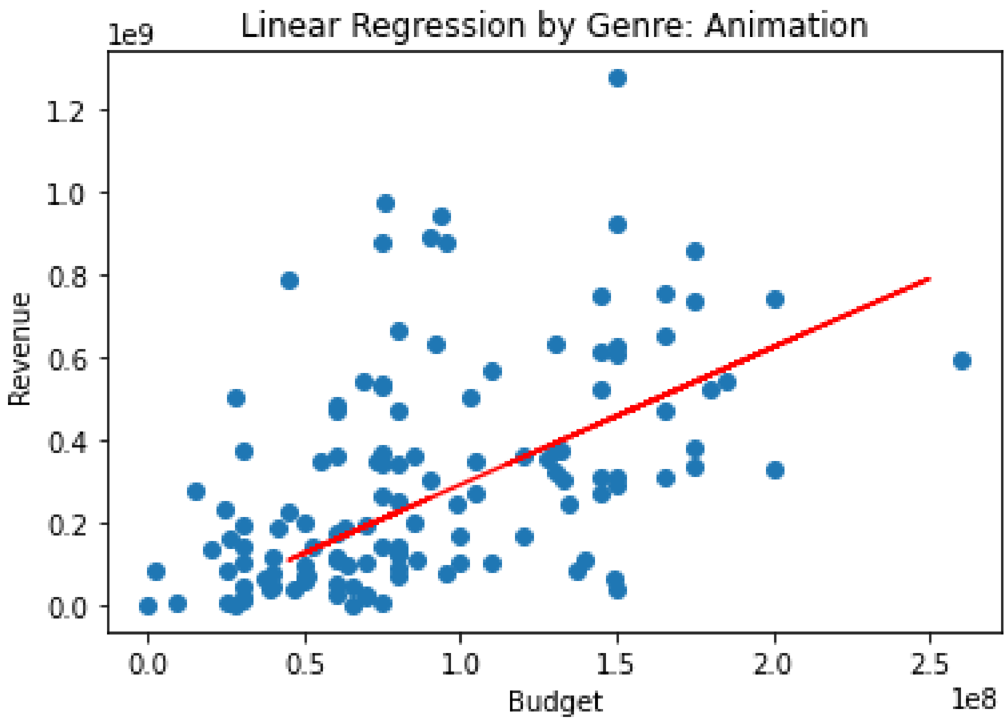
```
sum_RMSE = 0

kf = KFold(shuffle = True)
for train_index, test_index in kf.split(x):
    train, test = train_test_split(data)

    trainX = {}
    trainY = {}
    for ug in unique_genre:
        x = []
        y = []
        for l in range (0, len(train["genres"])):
            if (train["genres"].get(l) and train["revenue"].get(l) and ug in␣
 ↪train["genres"].get(l)):
                x.append(train["budget"].get(l))
                y.append(train["revenue"].get(l))
        trainX[ug] = x
        trainY[ug] = y

    testX = {}
    testY = {}
    for ug in unique_genre:
        x = []
        y = []
        for l in range (0, len(test["genres"])):
            if (test["genres"].get(l) and test["revenue"].get(l) and ug in␣
 ↪test["genres"].get(l)):
                x.append(test["budget"].get(l))
                y.append(test["revenue"].get(l))
        testX[ug] = x
        testY[ug] = y
```

[47]:
```
for g in unique_genre:
    if(testX.get(g) and testY.get(g) and trainX.get(g) and trainY.get(g)):
        # Gets all coordinates for x and y
        xTrain = np.array(trainX.get(g)).reshape((-1, 1))
        yTrain = np.array(trainY.get(g))
        xTest = np.array(testX.get(g)).reshape((-1, 1))
        yTest = np.array(testY.get(g))

        # Creates LinearRegression object
        m4 = LinearRegression()
        m4.fit(xTrain,yTrain)
        # Creates regression line based off coordinates
        pyplt.scatter(xTrain,yTrain)
        # Labels and organization
        pyplt.xlabel("Budget")
```

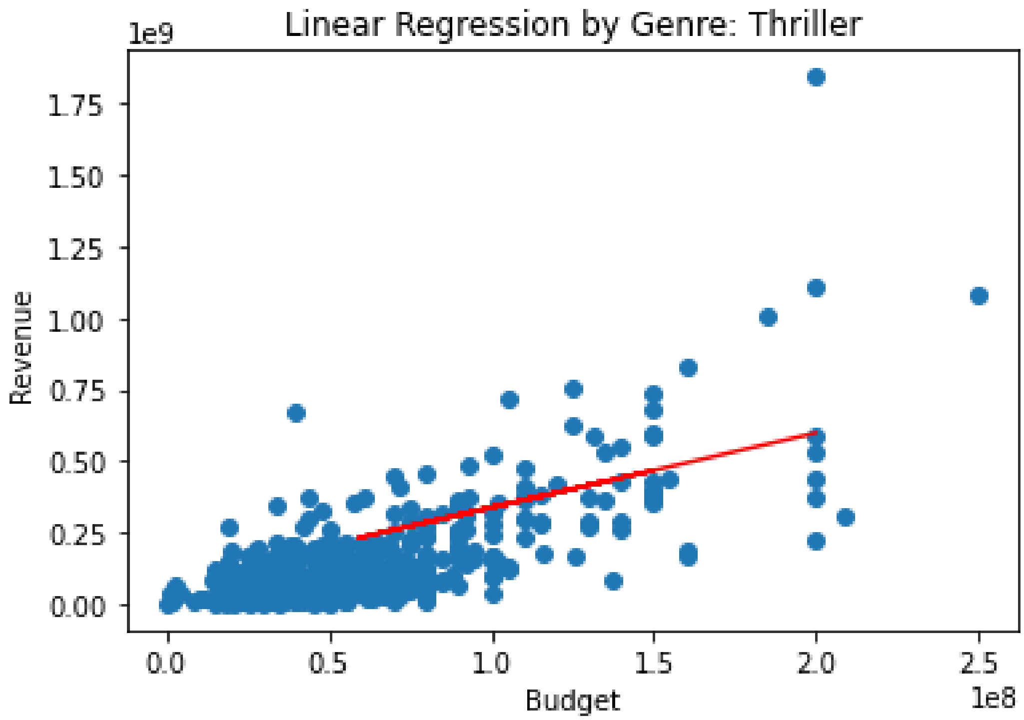
```
        pyplt.ylabel("Revenue")
        title = "Linear Regression by Genre: " + g
        pyplt.title(title)
        pyplt.show()

        # Gets variables for solving RMSE and MSE
        m4 = m4.fit(xTrain,yTrain)
        coef.append(m4.coef_[0])
        intercept.append(m4.intercept_)
        predictions = m4.predict(xTest)
        pyplt.plot(xTest, predictions, color="red")


        MSE = np.mean((yTest - predictions) ** 2)


        m4coef_ = np.array([np.mean(coef)])
        m4intercept_ = np.mean(intercept)
        print("The " + g + " movie linear regression model is: Revenue = " +␣
↪str(m4.coef_[0]) + " * Budget + " + str(m4.intercept_))

        # MSE_average = np.mean(MSE)
        print("Average MSE, for model #4:", MSE)
        print("Average RMSE:", MSE ** 0.5 / 1000000, 'million\n')
```



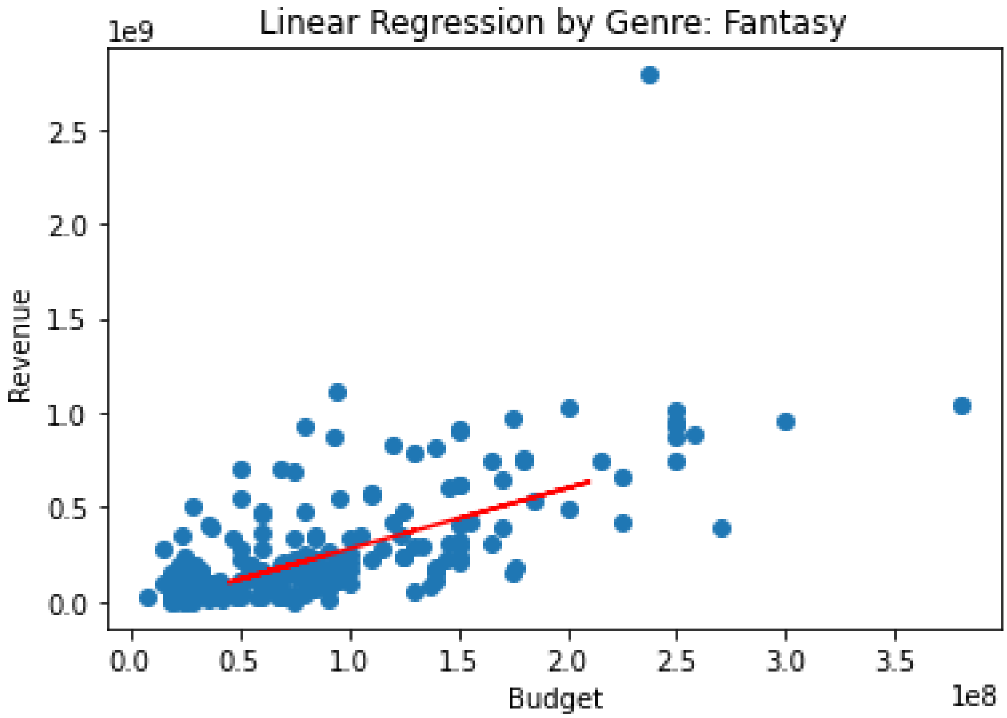
```
The Action movie linear regression model is: Revenue = 3.314173363316526 *
Budget + -40919878.84671065
Average MSE, for model #4: 5.216813556563179e+16
Average RMSE: 228.40344911062923 million
```



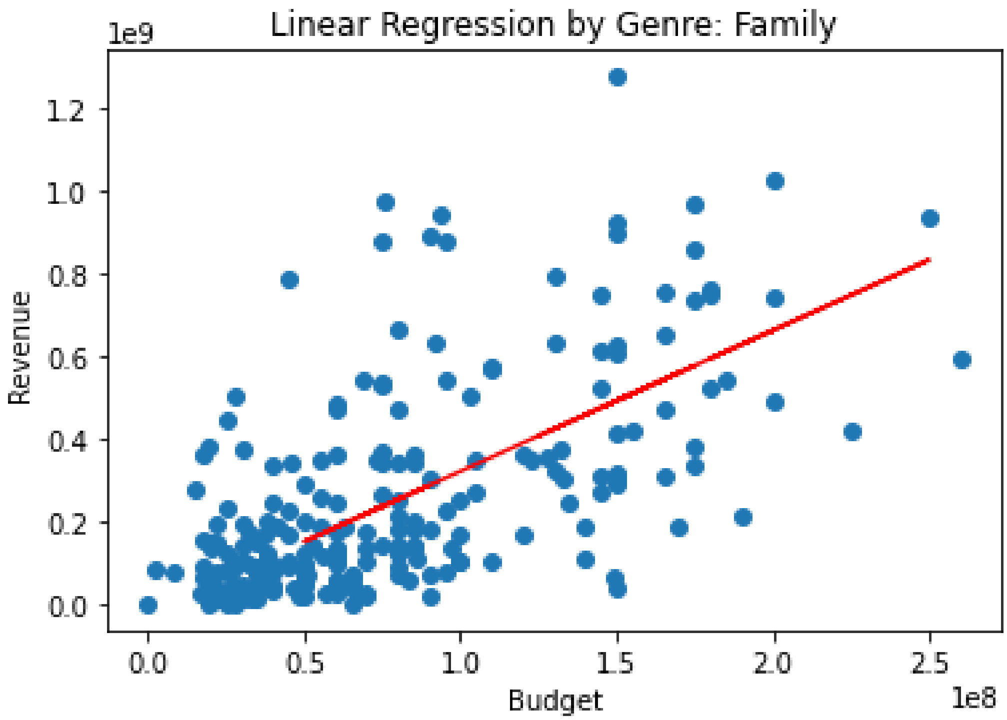

```
The Animation movie linear regression model is: Revenue = 2.602514406091154 *
Budget + 76734847.35849625
Average MSE, for model #4: 5.3915878082502856e+16
Average RMSE: 232.19792867832146 million
```



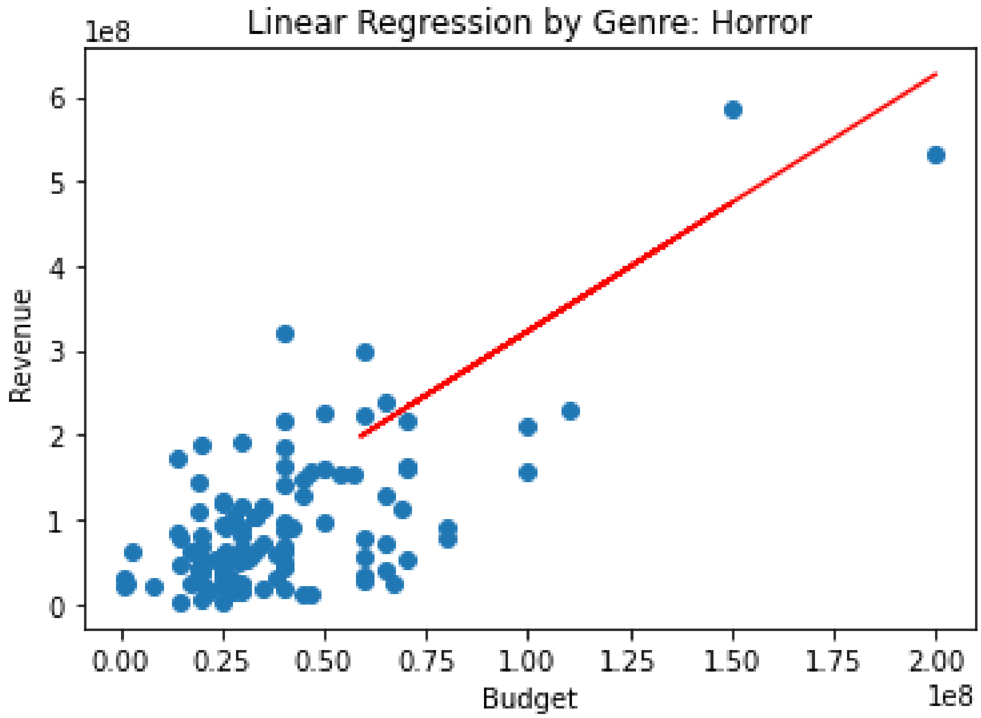

```
The Thriller movie linear regression model is: Revenue = 3.2209091154357883 *
Budget + -42898552.2609677
Average MSE, for model #4: 4.102203196224811e+16
Average RMSE: 202.53896405938318 million
```



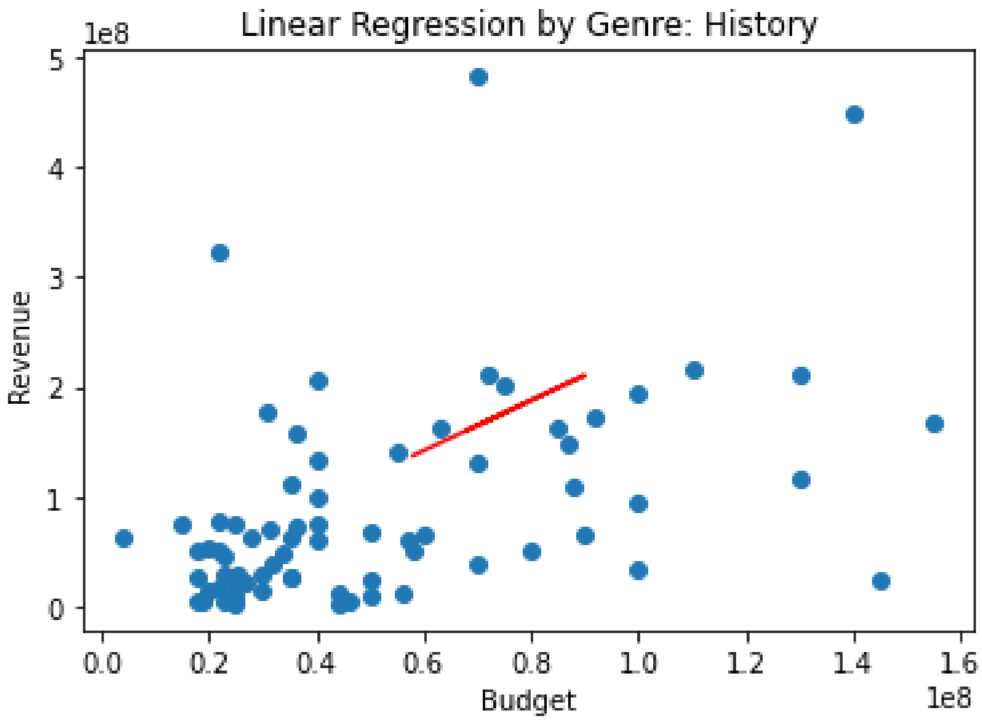

```
The Fantasy movie linear regression model is: Revenue = 3.4038737929643026 *
Budget + -19998190.504060686
Average MSE, for model #4: 4.946962003686822e+16
Average RMSE: 222.41767024422367 million
```



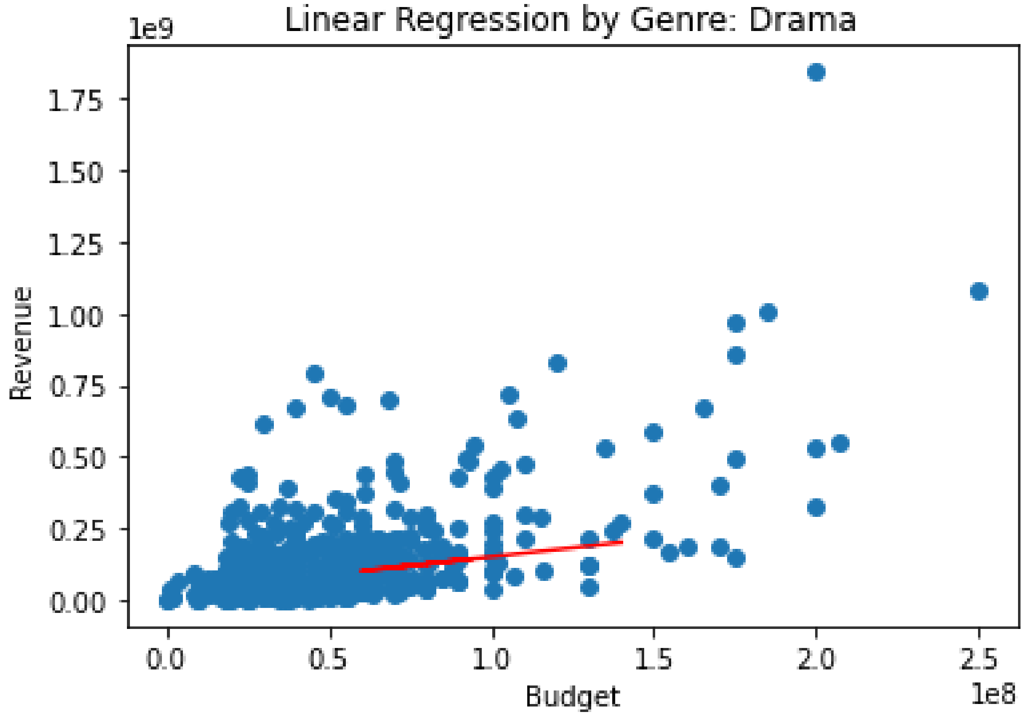

```
The Family movie linear regression model is: Revenue = 3.0380581540008285 *
Budget + 18824817.215588987
Average MSE, for model #4: 6.278223697822197e+16
Average RMSE: 250.56383812957122 million
```



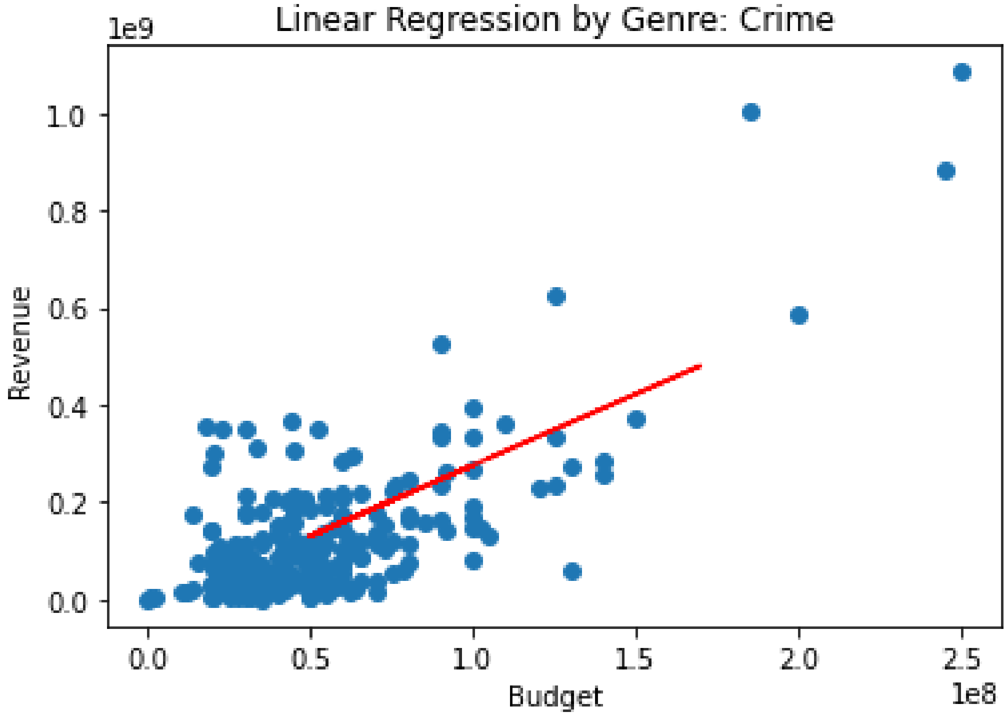

```
The Horror movie linear regression model is: Revenue = 2.2818014211086086 *
Budget + 5192815.014400631
Average MSE, for model #4: 1.164387974115751e+16
Average RMSE: 107.90681044844904 million
```



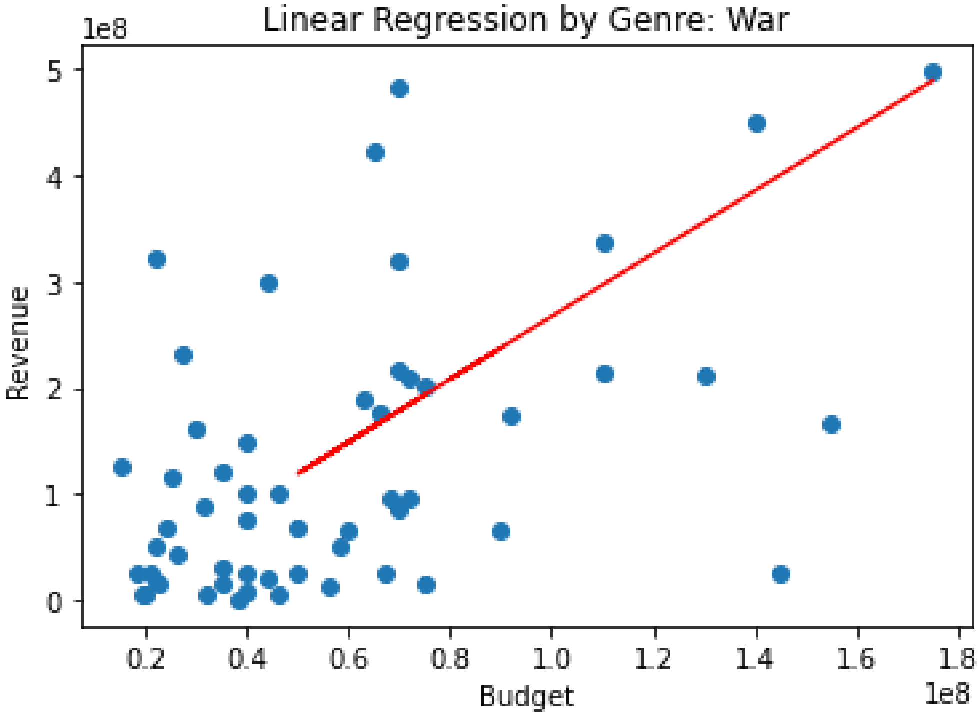

```
The History movie linear regression model is: Revenue = 1.2528559327528803 *
Budget + 24462645.912435286
Average MSE, for model #4: 1.1304554554301726e+16
Average RMSE: 106.32287879051114 million
```



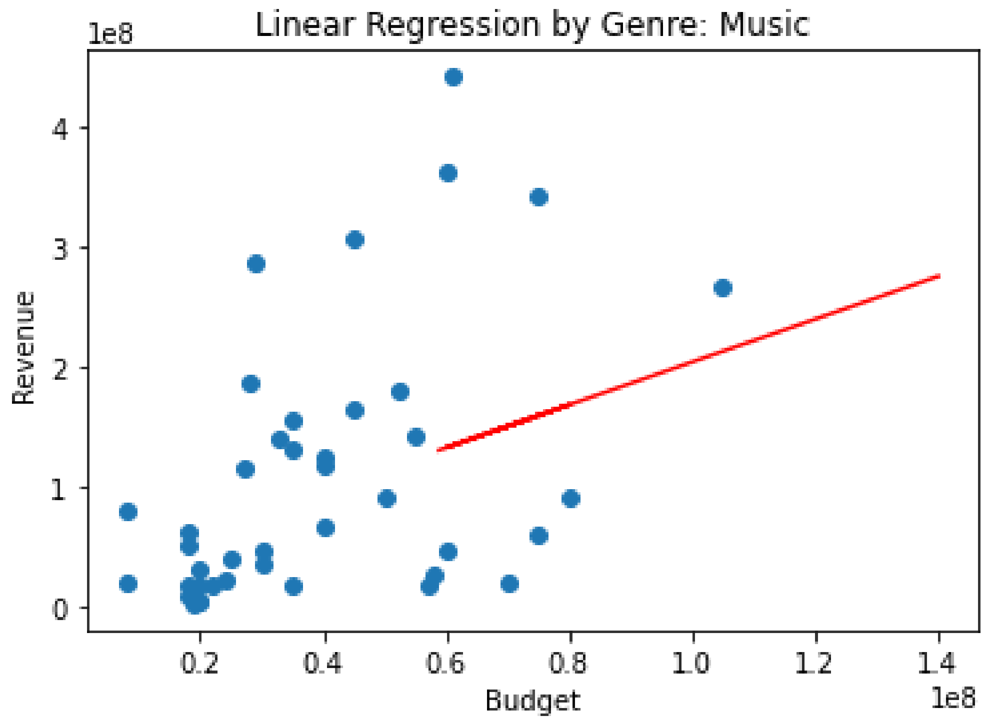
The Drama movie linear regression model is: Revenue = 2.9132461185327356 * Budget + -14999803.295148611
Average MSE, for model #4: 1.4503570621439594e+16
Average RMSE: 120.43077107383974 million



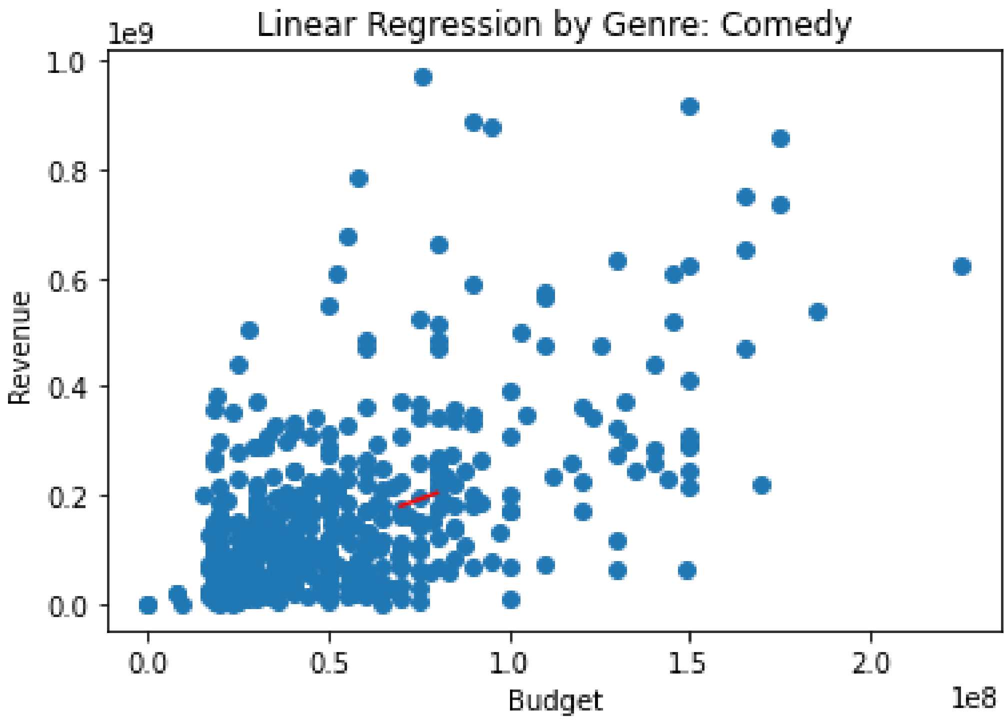
The Crime movie linear regression model is: Revenue = 2.956362534959314 * Budget + -28483207.188473344
Average MSE, for model #4: 9800733550954320.0
Average RMSE: 98.99865428860294 million



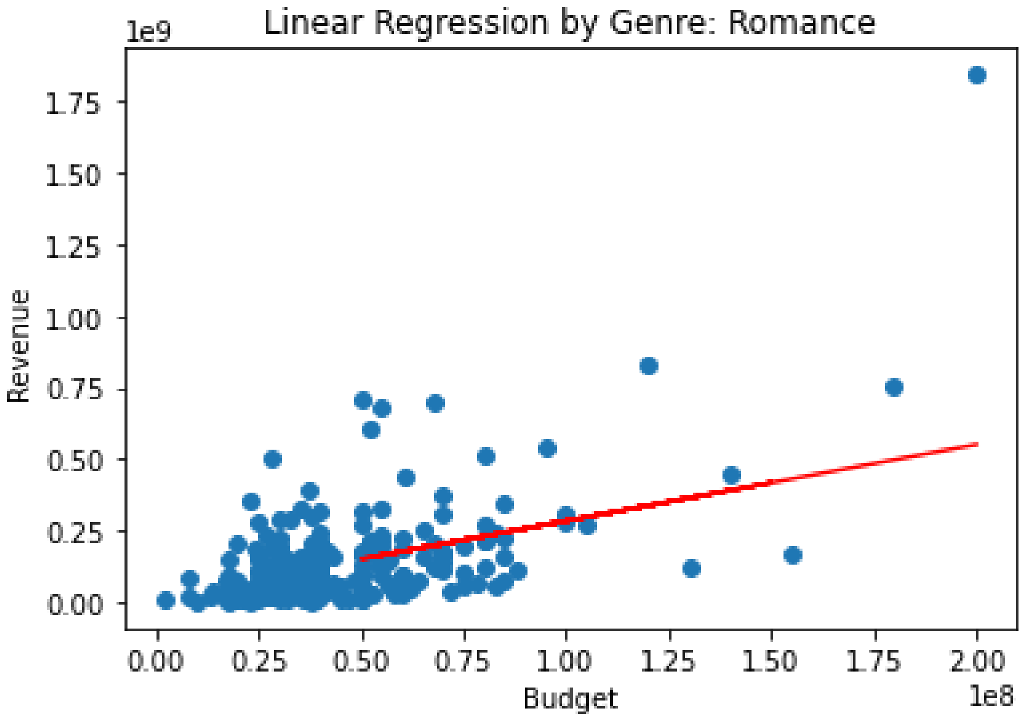
The War movie linear regression model is: Revenue = 1.788049214440682 * Budget + 24949808.40574637
Average MSE, for model #4: 2.356624095090857e+16
Average RMSE: 153.51299928966463 million



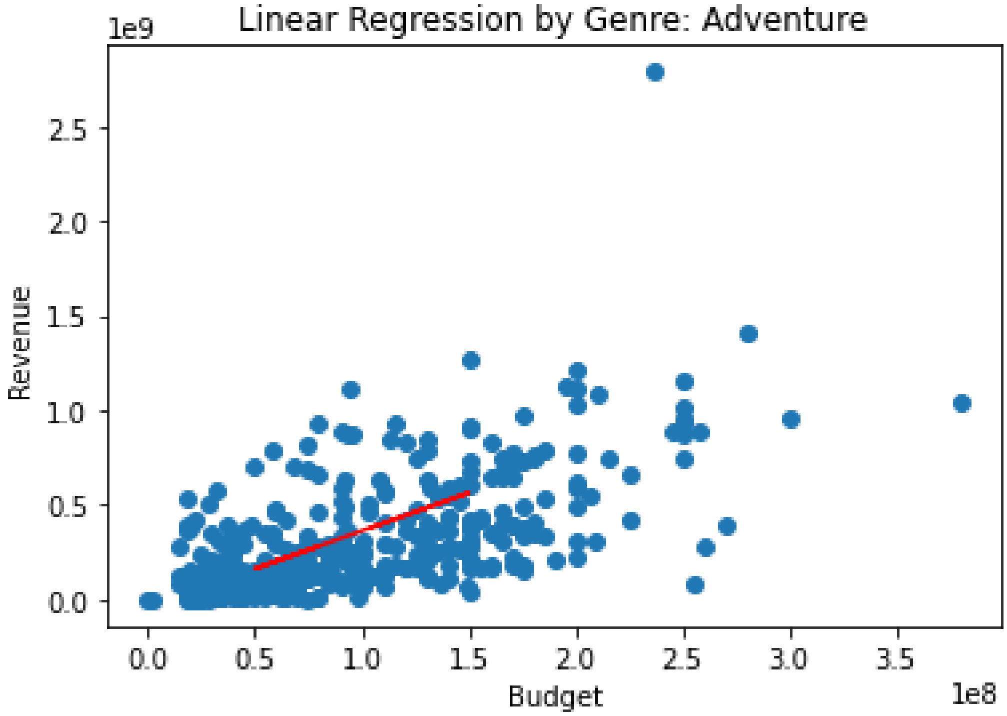

```
The Music movie linear regression model is: Revenue = 2.3999084129138053 *
Budget + 12120613.637696624
Average MSE, for model #4: 2.9027589867458628e+16
Average RMSE: 170.37485104163298 million
```



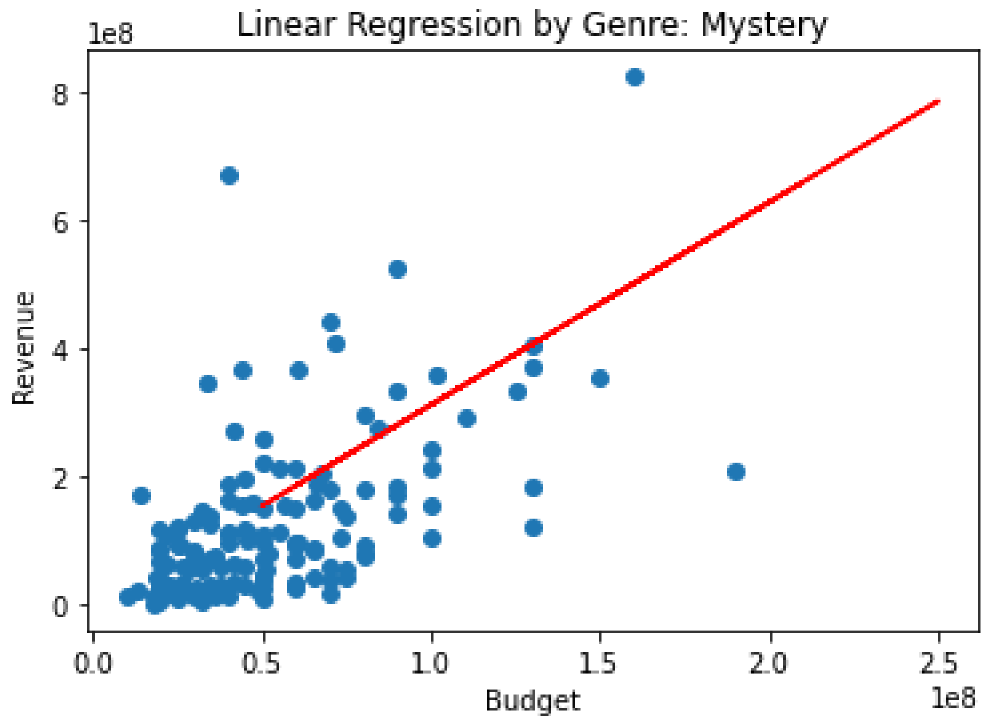

```
The Comedy movie linear regression model is: Revenue = 2.672671578764105 *
Budget + 15628496.681487858
Average MSE, for model #4: 3.8631840567794776e+16
Average RMSE: 196.549842451717 million
```



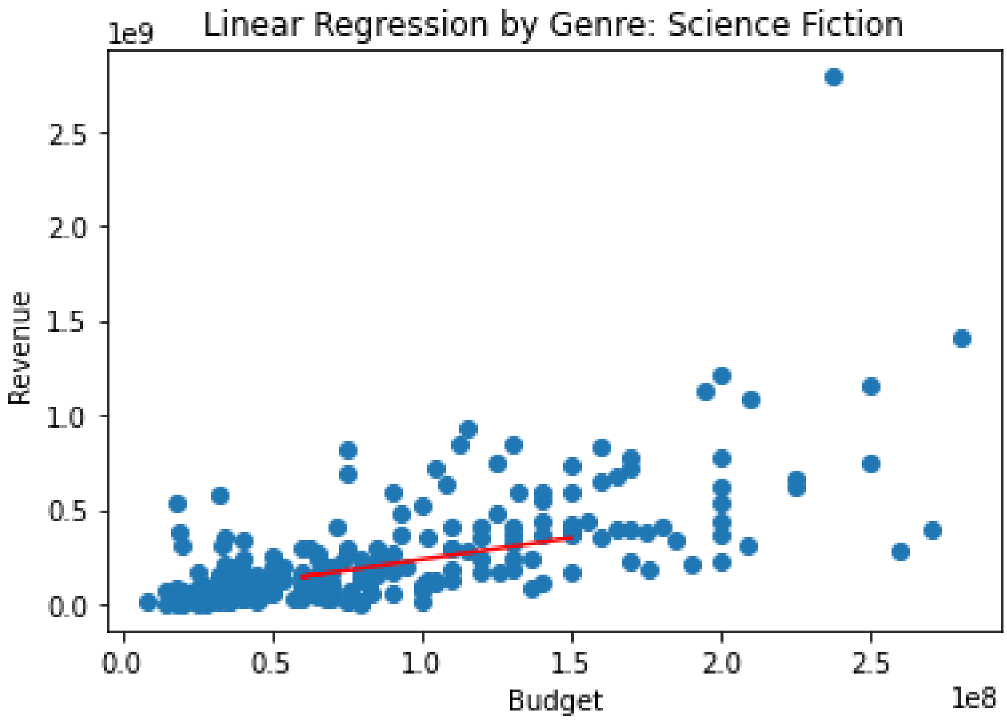

```
The Romance movie linear regression model is: Revenue = 4.010221917924004 *
Budget + -37987328.24089399
Average MSE, for model #4: 2.7095534500003828e+16
Average RMSE: 164.60721278244105 million
```



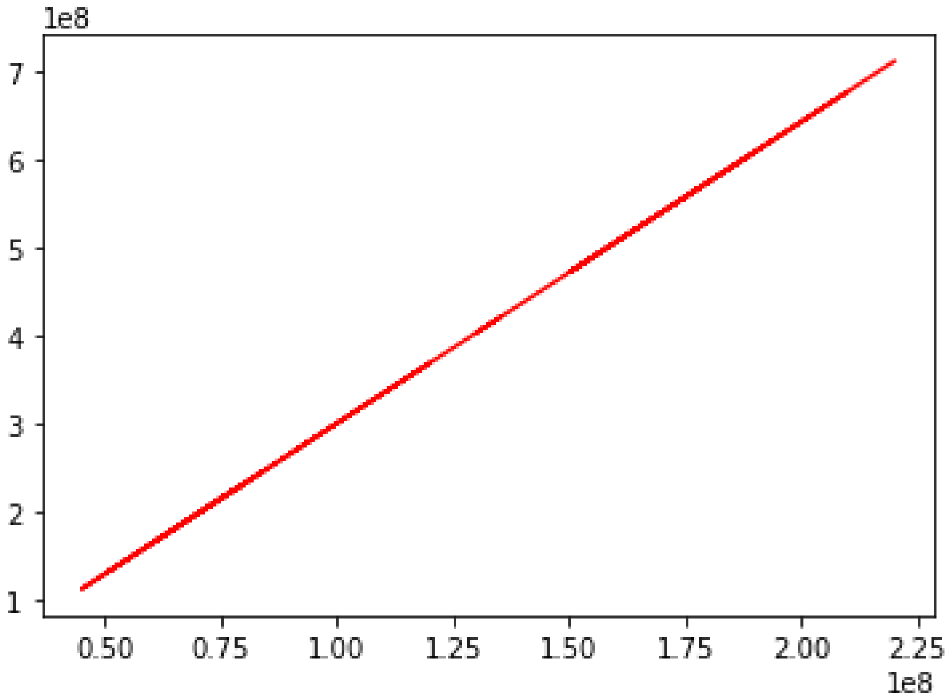
The Adventure movie linear regression model is: Revenue = 3.169573247703349 * Budget + -5117426.621167779
Average MSE, for model #4: 7.820002941286682e+16
Average RMSE: 279.6426816722848 million



The Mystery movie linear regression model is: Revenue = 2.2859071870746783 * Budget + 5818195.203875065
Average MSE, for model #4: 6.8048264709564e+16
Average RMSE: 260.86062314876887 million




```
The Science Fiction movie linear regression model is: Revenue =
3.430043757645975 * Budget + -42597506.745497316
Average MSE, for model #4: 8.57190893952868e+16
Average RMSE: 292.77822561674014 million
```



Here we have cross-validation using the applications of kfolding. As a result of kfolding and cross validation, the MSE has actually increased compared to without using cross validation drastically. Consequently, RMSE also increases since the RMSE is formulated from the MSE. In hindsight, a change that can improve this can be by removing outliers as stated before in addition to having a more diverse training set. Moreover, due to a bad test set the red line being the prediction line of the test set does not set a good prediction line due to the lack of selected movies in certain budgets.