Movie Revenue Prediction

June 30, 2021

0.1 Group Member: Jeffery 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
```

0.1.1 EDA

-

[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)
```

-

[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, the outliers that are caused by artifacts have to be removed. 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
```

-

[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
```

-

[11]:
```python
# 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]:
```python
# 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]:
```python
# Removes any genre with no popularity
new_all_info = {key:val for key, val in all_info.items() if val}
```

```
new_all_info
```

[13]:
```
{'Animation': [48.681969,
  89.186492,
  49.98659,
  59.995418,
  51.692953,
  125.114374,
  66.390712,
  39.744242,
  92.201962,
  36.167578,
  128.655964,
  44.041186,
  203.73459,
  62.341073,
  47.323228,
  67.263269,
  42.986467,
  65.677399,
  44.141021,
  165.125366,
  41.845878,
  29.332905,
  51.247321,
  12.362599,
  22.550135,
  38.73494,
  44.989192,
  35.601665,
  56.747978,
  61.788035,
  100.21391,
  26.074908,
  17.7735,
  44.765377,
  29.91353,
  84.366984,
  84.689648,
  63.473086,
  20.678787,
  89.938296,
  26.548594,
  68.757242,
  42.40024,
  27.990284,
  106.815545,
```

64.183321,
36.65422,
19.625972,
32.473628,
30.475297,
62.479574,
46.727941,
51.113717,
36.711378,
37.321848,
38.924136,
34.308098,
23.230851,
37.054554,
65.229868,
85.688789,
77.817571,
73.575118,
46.781182,
69.457898,
67.427755,
51.548589,
46.200042,
37.272385,
56.257411,
26.61951,
36.32705,
55.659988,
23.473004,
29.558157,
36.125715,
11.187696,
41.670544,
61.692197,
14.530946,
10.088006,
16.715631,
27.097932,
90.457886,
41.247402,
35.384423,
8.981427,
31.482872,
136.886704,
48.110909,
85.115058,
47.094369,

36.368839,
14.44381,
8.098369,
875.581305,
45.720894,
46.875375,
47.320801,
82.643036,
36.952268,
29.412262,
10.54544,
6.682006,
113.858273,
99.561972,
24.857079,
16.791058,
47.973995,
29.464852,
41.515702,
51.146047,
39.539045,
16.493576,
30.656345,
59.547928,
23.661071,
18.815442,
47.336034,
47.845589,
14.414937,
23.700759,
45.565918,
10.430832,
67.298732,
17.667826,
9.894346,
34.890999,
61.245957,
10.410933,
87.350802,
20.239061,
5.258726,
46.007991,
31.789029,
18.14218,
16.059443,
41.258956,
27.908783,

```
 6.293217,
 18.39217,
 14.051852,
 39.58676,
 4.098444,
 73.640445,
 27.870755,
 4.590277,
 30.380747,
 43.571612,
 55.806402,
 19.143721,
 10.694417,
 66.242737,
 92.982009,
 7.981116,
 14.726338,
 12.739721,
 5.753564,
 2.439184,
 49.549984,
 14.866351,
 15.341196,
 19.819893,
 9.829859,
 26.509175,
 12.300789,
 26.093043,
 60.732738,
 118.968562,
 9.244181,
 23.980329,
 6.459787,
 28.670193,
 11.103202,
 5.699058],
'Music': [62.479574,
 23.473004,
 14.530946,
 9.357734,
 31.160542,
 29.412262,
 23.515938,
 6.682006,
 48.356214,
 29.464852,
 21.547446,
```

47.336034,
7.027139,
57.374341,
65.90161,
8.59075,
17.047276,
6.803965,
21.743321,
14.199933,
17.659032,
10.794175,
11.60815,
10.73522,
29.775441,
4.098444,
32.217425,
13.788422,
43.571612,
12.113235,
8.999675,
71.496894,
35.580032,
61.76233,
44.714264,
26.576385,
6.889437,
25.03414,
2.753313,
13.960945,
21.815122,
43.057323,
4.748929,
26.509175,
7.902536,
23.982956,
0.709143,
8.237336,
7.716606,
13.032308,
0.068228,
31.82675,
18.492655,
6.961736,
7.871686,
28.973074,
7.180403,
20.786616,

```
 32.746486,
 5.700333,
 3.826685,
 0.335195,
 10.9071,
 13.832376,
 12.185204,
 29.660993,
 6.92205,
 6.735922,
 9.969624,
 11.880899,
 22.022228,
 45.477802,
 3.389249,
 6.703319,
 8.410688,
 36.319205,
 8.305998,
 20.452449,
 2.698185,
 6.18869,
 16.318608,
 17.611971],
'Family': [48.681969,
 98.885637,
 53.978602,
 78.530105,
 89.186492,
 46.985445,
 49.98659,
 59.995418,
 43.349855,
 51.692953,
 125.114374,
 66.390712,
 67.391328,
 92.201962,
 36.167578,
 32.319043,
 27.082182,
 128.655964,
 94.199316,
 110.620647,
 44.041186,
 130.311355,
 203.73459,
```

62.341073,
47.323228,
67.263269,
42.986467,
49.661984,
78.144395,
101.250416,
53.905592,
65.677399,
44.141021,
81.781591,
165.125366,
41.845878,
26.710398,
29.332905,
51.247321,
33.769336,
12.362599,
22.550135,
48.03528,
38.73494,
44.989192,
56.747978,
61.788035,
100.21391,
17.7735,
44.19092,
44.765377,
29.91353,
84.366984,
84.689648,
63.473086,
20.678787,
79.679601,
26.548594,
115.597753,
68.757242,
109.984351,
42.40024,
45.419668,
27.990284,
17.060695,
106.815545,
64.183321,
20.344364,
48.780039,
36.65422,

19.625972,
32.473628,
30.475297,
62.479574,
132.397737,
46.727941,
51.113717,
30.863434,
31.586215,
36.711378,
37.321848,
38.924136,
23.230851,
37.054554,
65.229868,
101.187052,
85.688789,
21.436682,
77.817571,
61.121717,
73.575118,
46.781182,
69.457898,
67.427755,
51.548589,
16.302378,
40.262261,
46.200042,
3.593349,
56.257411,
29.21998,
9.895061,
26.61951,
32.044191,
48.126297,
11.77398,
36.32705,
55.659988,
14.132583,
23.473004,
29.558157,
23.531721,
11.50396,
36.125715,
11.187696,
41.670544,
27.46264,

17.455024,
61.692197,
14.530946,
10.088006,
16.715631,
11.356079,
27.097932,
90.457886,
41.247402,
8.981427,
31.482872,
136.886704,
48.110909,
85.115058,
31.160542,
13.25753,
47.094369,
36.368839,
14.44381,
875.581305,
45.720894,
13.543053,
46.875375,
8.884318,
21.462935,
47.320801,
82.643036,
36.952268,
33.648404,
29.412262,
48.032734,
10.54544,
6.682006,
113.858273,
99.561972,
14.646884,
18.281798,
33.439187,
15.194239,
20.415572,
24.857079,
16.791058,
30.951203,
47.973995,
29.464852,
18.106021,
18.735024,

41.515702,
51.146047,
4.280619,
22.501041,
39.539045,
16.493576,
30.656345,
59.547928,
6.80692,
23.661071,
18.815442,
18.676291,
27.018886,
6.248309,
35.4763,
47.336034,
47.845589,
19.830132,
32.835777,
14.414937,
39.882276,
23.700759,
14.808304,
9.27875,
45.565918,
10.430832,
67.298732,
38.629763,
17.667826,
23.935053,
9.894346,
12.050755,
34.890999,
12.31101,
12.25622,
61.245957,
21.312186,
10.410933,
28.964162,
53.090422,
30.002907,
30.816341,
17.047276,
87.350802,
8.99269,
51.742065,
20.239061,

13.707183,
5.258726,
46.007991,
31.789029,
14.587367,
43.345252,
21.998734,
45.769562,
24.287306,
18.14218,
16.059443,
41.258956,
16.282962,
27.908783,
24.085144,
41.258487,
23.465539,
14.353279,
6.63839,
6.967857,
6.293217,
21.543389,
11.041659,
18.39217,
13.707843,
6.889633,
22.71592,
14.051852,
10.304172,
39.58676,
16.901954,
14.642055,
10.651202,
4.098444,
73.640445,
40.176265,
27.870755,
13.788422,
4.590277,
34.36645,
30.380747,
19.143721,
10.694417,
8.386432,
92.982009,
7.981116,
18.650748,

```
22.14595,
22.074306,
14.639324,
1.767479,
47.862016,
5.650125,
14.788073,
31.250383,
5.894163,
4.475007,
30.884535,
12.739721,
15.505137,
40.646102,
24.356483,
10.81439,
27.683874,
13.288177,
7.538513,
5.753564,
25.03414,
6.589193,
2.753313,
2.439184,
8.842722,
16.234053,
11.263361,
20.699212,
13.410919,
4.365196,
10.207065,
81.487685,
12.25264,
13.590677,
15.341196,
19.819893,
9.829859,
14.136476,
8.265317,
12.300789,
4.100942,
11.090254,
7.716606,
76.603233,
118.968562,
0.068228,
20.716023,
```

```
 16.027955,
 14.329891,
 18.492655,
 9.191187,
 4.492859,
 23.980329,
 20.786616,
 7.864011,
 43.217089,
 2.186927,
 21.095489,
 9.125594,
 11.927667,
 44.533612,
 3.334804,
 6.459787,
 13.13724,
 6.735922,
 9.969624,
 4.580081,
 17.371327,
 28.670193,
 0.903529,
 11.103202,
 14.779537,
 6.703319,
 4.623609,
 16.931846,
 1.081822,
 56.105798,
 8.932837,
 8.305998,
 5.699058,
 31.974359,
 0.518056],
'Romance': [100.025899,
 61.196071,
 62.169881,
 110.620647,
 60.269279,
 34.20669,
 39.019229,
 99.687084,
 11.137655,
 62.479574,
 32.389353,
 41.426678,
```

18.325897,
101.187052,
20.913852,
24.100863,
29.21998,
44.967453,
16.479851,
16.595874,
17.783361,
41.176631,
34.070054,
16.939441,
19.500893,
23.531721,
37.02665,
18.264635,
27.46264,
21.891078,
9.357734,
50.561849,
59.248437,
61.437586,
31.160542,
23.004607,
24.353302,
19.143321,
21.789615,
13.543053,
11.215702,
20.230535,
27.299301,
48.032734,
23.515938,
107.069763,
25.877794,
31.39165,
48.356214,
28.540267,
59.634029,
38.994297,
4.137237,
11.056763,
13.218008,
32.829538,
24.153567,
26.676487,
12.867352,

13.630049,
22.620125,
13.155999,
22.736038,
45.514031,
13.43573,
6.248309,
7.739276,
9.061866,
138.133331,
29.97224,
20.28119,
34.576913,
36.194802,
19.738215,
15.064058,
21.272734,
18.927463,
15.274137,
22.863516,
23.962109,
22.894898,
18.600367,
57.374341,
26.251217,
30.780599,
25.41929,
24.444186,
94.815867,
26.60772,
27.039636,
37.885022,
18.121404,
23.854701,
50.163785,
37.990705,
12.050755,
34.890999,
28.803802,
41.362082,
13.298205,
6.22577,
42.933027,
7.44268,
8.189406,
40.721373,
19.348007,

```
8.59075,
16.000178,
18.102572,
7.226115,
6.067212,
15.401855,
30.002907,
10.411996,
37.073127,
10.731574,
8.99269,
20.360549,
5.361699,
18.65632,
28.969151,
33.238296,
6.420434,
13.244365,
36.670399,
36.238968,
98.755657,
26.253357,
45.022964,
22.853006,
37.776566,
29.854773,
21.021809,
34.519636,
14.231899,
29.88736,
19.574774,
17.659032,
24.313684,
9.965073,
22.491388,
10.433667,
19.372805,
16.059443,
8.980846,
73.720244,
8.303696,
37.492928,
2.817364,
18.256028,
22.332927,
16.801218,
6.454634,
```

6.760922,
27.009723,
46.078371,
127.084938,
29.652536,
70.398356,
83.893257,
3.74263,
29.381878,
40.526587,
11.120167,
18.38292,
14.029012,
22.636908,
8.737058,
14.699342,
10.10364,
16.736734,
16.343317,
3.746806,
16.782471,
9.07278,
2.877488,
5.516207,
9.133747,
24.606012,
41.946523,
29.775441,
33.923786,
15.039407,
17.107065,
6.201404,
5.347936,
13.611544,
36.432598,
26.780961,
40.176265,
55.109138,
30.695228,
13.788422,
3.845231,
26.952727,
33.704181,
43.742241,
12.967137,
50.670253,
41.569541,

```
43.753585,
10.81051,
29.489274,
33.369305,
43.571612,
82.052056,
25.060342,
32.26789,
20.355458,
6.246199,
34.421696,
20.236951,
4.352422,
9.301562,
19.106154,
3.362045,
8.386432,
35.580032,
18.165957,
8.93941,
92.982009,
14.530803,
37.729059,
36.58383,
22.074306,
28.397048,
16.785866,
22.527211,
12.576381,
28.227984,
38.473732,
31.792964,
21.988861,
21.219292,
14.831402,
33.93914,
26.576385,
35.310712,
28.727731,
38.688822,
26.811541,
25.838073,
5.625632,
21.729434,
10.81439,
8.870614,
19.402816,
```

15.960916,
7.946769,
3.593165,
30.058263,
14.480628,
15.237142,
25.03414,
5.771479,
7.701041,
8.630825,
53.682367,
8.976738,
5.494047,
6.618149,
41.967005,
57.110486,
17.606087,
12.887673,
13.960945,
16.668854,
14.866351,
15.013386,
5.683281,
63.599973,
10.13559,
8.982335,
13.590677,
4.748929,
2.986286,
47.51363,
53.161905,
2.003047,
11.203335,
23.982956,
56.481487,
19.962713,
16.176976,
17.237143,
17.145287,
27.256849,
14.370111,
8.244324,
5.596835,
19.220789,
5.133744,
0.709143,
45.742217,

```
10.218307,
11.267676,
27.494037,
17.952817,
21.959742,
11.766564,
16.665841,
85.30318,
20.716023,
16.027955,
12.243887,
18.492655,
13.472365,
11.725805,
9.633711,
12.313143,
1.876811,
10.47644,
28.973074,
20.786616,
23.425431,
24.054062,
32.746486,
3.960098,
5.787891,
36.960953,
25.112567,
4.358255,
20.898711,
18.236284,
18.430419,
29.943316,
41.083914,
7.854989,
15.976335,
8.8321,
32.758254,
9.119797,
13.832464,
9.400404,
20.818567,
58.900647,
45.742254,
3.803495,
22.838834,
61.182331,
6.857923,
```

10.194097,
44.533612,
5.164281,
2.027393,
30.669913,
15.355589,
7.809701,
16.629173,
3.033471,
10.9071,
8.074807,
21.857387,
30.983397,
12.185204,
52.744331,
41.493042,
59.258969,
16.742972,
45.473484,
3.650857,
6.772615,
4.536745,
4.66723,
11.942862,
0.900442,
24.210698,
44.135568,
9.982885,
9.969624,
11.828068,
54.550275,
6.130096,
1.750054,
10.573191,
17.632446,
43.444135,
4.267466,
19.664699,
13.349012,
35.268557,
15.817096,
7.497306,
60.767168,
6.035152,
19.195399,
45.477802,
40.631671,

```
 3.389249,
 6.703319,
 18.422327,
 74.358971,
 16.931846,
 64.610671,
 0.700792,
 1.623338,
 8.290636,
 12.664787,
 5.841285,
 12.854853,
 33.283554,
 23.449703,
 39.337949,
 29.543331,
 7.722021,
 7.063053,
 9.371811,
 33.282794,
 1.187671,
 9.705589,
 3.246903,
 4.519523,
 12.792446,
 8.9876,
 5.882208,
 20.452449,
 9.559737,
 11.350382,
 3.079862,
 30.863082,
 6.18869,
 14.905357,
 0.518056,
 3.007816,
 3.988432,
 10.733761,
 4.570043,
 9.805551,
 1.617943,
 3.588529,
 17.151293,
 19.747809,
 7.623979,
 7.720692],
'Comedy': [52.035179,
```

49.98659,
59.995418,
125.114374,
22.57178,
92.201962,
40.748915,
27.082182,
128.655964,
44.041186,
203.73459,
62.341073,
35.580815,
42.986467,
53.905592,
65.677399,
81.781591,
41.845878,
26.710398,
50.306728,
29.332905,
22.550135,
66.21806,
91.332849,
56.747978,
25.159168,
100.21391,
24.855701,
21.605568,
17.7735,
29.91353,
84.366984,
84.689648,
63.473086,
38.068736,
115.597753,
68.757242,
39.448066,
42.40024,
45.419668,
33.616115,
27.990284,
11.137655,
44.906918,
106.815545,
64.183321,
20.344364,
143.350376,

39.873791,
48.780039,
44.635452,
22.845143,
31.703608,
36.65422,
32.473628,
22.643776,
30.475297,
32.389353,
24.399642,
51.113717,
95.007934,
19.38841,
12.092241,
23.230851,
19.224754,
18.325897,
37.054554,
65.229868,
104.121555,
73.575118,
35.814765,
46.781182,
69.457898,
43.348022,
40.20395,
43.192048,
40.597344,
140.849495,
16.302378,
51.535701,
46.200042,
24.939295,
37.272385,
56.257411,
29.21998,
9.895061,
36.743324,
44.967453,
16.479851,
24.107835,
32.044191,
48.126297,
38.885195,
17.783361,
31.435539,

```
41.176631,
13.661289,
11.77398,
109.684788,
58.040149,
36.32705,
14.132583,
26.065735,
23.473004,
45.589568,
23.496054,
16.939441,
19.500893,
29.558157,
23.531721,
37.02665,
11.50396,
36.125715,
18.264635,
11.187696,
41.670544,
27.46264,
61.692197,
40.138009,
44.090535,
16.715631,
11.356079,
41.247402,
22.132418,
8.981427,
136.886704,
85.115058,
61.437586,
31.160542,
38.864027,
57.753914,
13.25753,
18.70031,
47.094369,
18.61472,
23.004607,
48.744209,
14.209329,
36.368839,
24.353302,
10.605084,
8.098369,
```

875.581305,
21.789615,
20.199057,
12.599836,
13.543053,
46.875375,
21.462935,
47.320801,
82.643036,
20.230535,
36.952268,
33.648404,
29.412262,
14.780634,
14.086292,
13.267631,
23.515938,
10.54544,
10.004564,
68.734513,
12.948627,
27.11589,
11.023691,
31.39165,
99.561972,
14.646884,
28.540267,
22.977983,
43.865294,
18.281798,
31.349143,
33.439187,
59.634029,
38.994297,
16.803457,
40.137264,
12.421504,
11.056763,
13.218008,
11.262655,
16.791058,
32.829538,
30.951203,
47.973995,
56.488027,
18.106021,
42.881415,

18.735024,
24.153567,
41.515702,
26.676487,
18.831023,
41.430245,
4.280619,
28.848842,
22.501041,
39.539045,
29.876507,
30.656345,
15.142701,
59.547928,
25.293538,
13.630049,
6.80692,
23.661071,
22.620125,
21.08952,
18.676291,
14.84856,
52.417529,
514.569956,
45.351617,
13.43573,
6.248309,
7.739276,
14.403696,
1.551497,
21.723897,
14.113066,
35.4763,
138.133331,
47.336034,
29.97224,
20.28119,
34.576913,
36.194802,
38.06168,
19.830132,
19.738215,
22.787667,
29.498304,
25.400935,
32.835777,
14.414937,

39.882276,
39.395881,
8.802525,
23.700759,
14.808304,
9.27875,
22.894898,
18.600367,
26.251217,
30.780599,
25.41929,
15.98212,
32.015477,
24.444186,
39.001682,
10.430832,
67.298732,
27.039636,
68.87691,
37.885022,
35.537397,
38.125008,
18.121404,
23.854701,
52.380982,
43.849351,
45.318703,
38.616744,
65.90161,
38.629763,
18.806031,
37.990705,
17.667826,
16.251204,
9.894346,
12.050755,
34.890999,
39.920035,
20.333953,
33.98637,
28.803802,
13.298205,
12.25622,
6.22577,
25.237969,
21.408222,
21.312186,

```
37.253774,
6.397388,
2.808428,
31.02233,
34.973086,
18.60049,
19.348007,
16.000178,
19.525659,
7.859226,
7.226115,
10.410933,
15.401855,
26.721386,
28.964162,
36.585509,
32.624355,
53.090422,
30.002907,
31.305336,
10.411996,
30.816341,
35.376971,
17.047276,
26.79781,
37.073127,
87.350802,
10.731574,
8.99269,
51.742065,
44.120328,
6.156015,
28.029015,
20.239061,
13.707183,
6.803965,
5.201688,
35.6585,
5.258726,
18.65632,
62.714026,
28.969151,
31.789029,
33.238296,
6.420434,
37.654688,
14.587367,
```

```
19.438503,
74.168016,
7.663515,
9.419726,
36.238968,
43.345252,
21.998734,
26.516624,
21.661028,
26.253357,
32.549282,
45.769562,
36.301202,
67.767785,
53.114573,
25.136082,
25.391899,
29.854773,
17.61292,
21.021809,
52.571674,
34.519636,
17.869443,
14.231899,
23.088571,
82.185281,
18.14218,
19.574774,
17.659032,
43.151268,
15.283444,
72.641296,
3.415534,
9.965073,
22.491388,
4.842626,
4.516953,
2.410546,
41.258956,
8.980846,
7.222985,
73.720244,
15.219263,
15.590001,
13.119133,
27.908783,
22.164202,
```

```
21.362949,
18.256028,
22.332927,
41.258487,
11.60815,
6.760922,
27.009723,
31.941347,
14.353279,
25.825771,
46.078371,
16.865782,
100.348805,
33.926473,
29.652536,
1.068453,
82.21166,
18.316151,
36.172182,
21.543389,
50.309843,
83.893257,
34.005726,
72.774941,
3.74263,
11.041659,
18.39217,
40.526587,
50.179171,
31.957947,
38.429648,
13.707843,
14.503568,
11.120167,
18.38292,
14.029012,
6.889633,
8.737058,
14.699342,
23.835055,
22.71592,
19.088655,
11.195593,
18.262017,
16.736734,
13.668633,
10.73522,
```

10.304172,
16.782471,
20.27704,
16.832312,
3.276912,
24.389194,
20.908494,
16.901954,
14.642055,
17.107065,
38.21708,
18.84332,
15.901137,
10.032626,
6.201404,
26.9385,
4.098444,
13.611544,
74.417456,
15.895661,
47.753598,
15.585138,
32.217425,
34.86859,
36.432598,
73.640445,
41.156217,
37.179547,
34.391792,
40.176265,
10.704596,
27.870755,
29.415229,
30.695228,
13.788422,
4.590277,
19.925026,
43.742241,
34.36645,
12.967137,
50.670253,
11.409124,
33.385429,
8.584398,
41.569541,
43.753585,
28.624038,

```
30.380747,
8.058109,
14.585968,
32.26789,
10.608998,
12.821329,
12.113235,
20.355458,
15.67747,
4.52363,
6.246199,
27.71063,
2.850896,
2.77255,
20.236951,
14.729386,
11.374306,
4.352422,
18.27083,
3.362045,
10.694417,
4.952735,
66.242737,
71.496894,
8.085872,
18.165957,
8.93941,
20.55509,
12.653831,
92.982009,
19.237254,
24.917665,
57.518472,
14.949395,
24.13081,
28.043508,
14.530803,
38.0958,
18.650748,
15.794946,
25.039383,
22.14595,
22.074306,
4.436792,
14.639324,
1.767479,
12.921594,
```

28.397048,
16.785866,
22.527211,
6.248273,
44.714264,
28.227984,
11.678561,
14.788073,
9.670031,
8.31653,
31.250383,
24.345651,
38.473732,
5.411154,
41.832392,
21.988861,
39.742009,
10.925618,
21.219292,
3.314094,
30.884535,
12.739721,
15.505137,
28.468,
35.310712,
28.727731,
4.340616,
14.604911,
38.688822,
29.342744,
24.356483,
6.086083,
17.17372,
25.838073,
5.625632,
21.729434,
10.81439,
27.683874,
14.340095,
6.889437,
19.428994,
6.081772,
17.329269,
20.561682,
13.288177,
19.212552,
29.365851,

8.870614,
15.960916,
25.848003,
3.707511,
39.784026,
29.796394,
7.946769,
5.073536,
8.186936,
20.011435,
3.593165,
15.279021,
15.237142,
23.441135,
24.70166,
16.146572,
5.771479,
27.326466,
20.69219,
7.701041,
6.589193,
3.113009,
5.566203,
8.630825,
23.343839,
57.300674,
23.839127,
33.872182,
27.33256,
8.976738,
5.494047,
39.890798,
57.110486,
16.234053,
20.699212,
13.786075,
17.772518,
10.259461,
30.114383,
12.887673,
13.410919,
21.865021,
39.006187,
16.668854,
15.013386,
20.410354,
10.207065,

5.683281,
63.599973,
10.13559,
5.815881,
4.189378,
81.487685,
25.28436,
8.351385,
21.815122,
20.99174,
12.25264,
13.590677,
21.657563,
43.057323,
28.703007,
40.903931,
4.748929,
2.986286,
42.512296,
10.098722,
6.064638,
2.003047,
11.203335,
17.165039,
20.311684,
26.509175,
36.819044,
7.902536,
17.971883,
23.982956,
19.962713,
55.964605,
16.176976,
17.237143,
4.100942,
12.309715,
17.145287,
11.090254,
8.244324,
13.658985,
5.596835,
16.238961,
6.397042,
19.220789,
14.031377,
11.352693,
9.36146,

6.102663,
0.709143,
45.742217,
2.605374,
12.232048,
7.716606,
10.218307,
5.65327,
10.617236,
28.497242,
25.28132,
76.603233,
27.494037,
18.47518,
17.952817,
21.959742,
11.766564,
26.441669,
44.511363,
16.665841,
36.471231,
85.30318,
20.716023,
29.487784,
25.43954,
9.832326,
6.501815,
26.58873,
36.15716,
16.027955,
14.329891,
36.423717,
12.243887,
89.811154,
34.694216,
4.504377,
9.191187,
9.633711,
24.548633,
33.436092,
23.980329,
70.668923,
23.425431,
14.304444,
7.210095,
23.491962,
12.709227,

```
32.746486,
3.960098,
5.787891,
7.864011,
9.349386,
10.74648,
35.777067,
18.430419,
14.759948,
29.943316,
41.083914,
15.976335,
38.623178,
8.8321,
9.119797,
32.257414,
3.021835,
9.455596,
0.830596,
5.700333,
9.400404,
18.923181,
29.843818,
2.186927,
20.818567,
17.638286,
21.095489,
22.838834,
27.165222,
58.553213,
2.005111,
9.125594,
13.581966,
11.927667,
31.565117,
23.423082,
6.857923,
10.194097,
44.533612,
7.424798,
5.164281,
3.334804,
2.027393,
13.237835,
18.268009,
3.033471,
49.230089,
```

8.074807,
7.034752,
25.329493,
13.832376,
7.446069,
21.857387,
32.690038,
8.583217,
12.185204,
52.744331,
29.660993,
45.473484,
22.409117,
5.102258,
6.772615,
6.733485,
2.238646,
4.536745,
4.66723,
7.487843,
16.618874,
32.306455,
18.362333,
25.316146,
43.791745,
11.828068,
54.550275,
4.580081,
6.425528,
27.546914,
6.764437,
1.750054,
6.170139,
4.742537,
35.522027,
17.632446,
1.647335,
12.321302,
30.952377,
19.664699,
15.817096,
7.497306,
26.396191,
6.546037,
60.767168,
11.103202,
15.26213,

6.035152,
10.297189,
41.249004,
40.631671,
26.219598,
14.779537,
6.703319,
49.070321,
4.623609,
11.510281,
16.931846,
1.081822,
16.183746,
0.700792,
4.330428,
4.770195,
1.623338,
2.993939,
32.523573,
45.072136,
56.805781,
11.227087,
19.231507,
7.559192,
14.282314,
3.495758,
4.12064,
2.259537,
7.722021,
7.063053,
9.371811,
1.187671,
4.519523,
12.792446,
0.037073,
10.892398,
14.754281,
8.9876,
5.882208,
8.932837,
26.598884,
25.929893,
21.258991,
5.699058,
7.499639,
4.928777,
20.452449,

```
 14.543435,
 3.290639,
 9.559737,
 11.350382,
 18.289303,
 17.592299,
 4.207456,
 5.647538,
 18.366362,
 6.736274,
 5.671691,
 14.038512,
 35.350303,
 30.863082,
 14.105392,
 14.905357,
 0.518056,
 16.318608,
 10.409155,
 19.263477,
 3.988432,
 14.979375,
 4.570043,
 9.805551,
 0.666218,
 1.728816,
 1.617943,
 19.978021,
 9.337388,
 14.562991,
 19.747809,
 7.720692,
 3.951041,
 18.514292,
 34.022209,
 2.062087,
 8.738737],
'Action': [150.437577,
 139.082615,
 107.376788,
 112.31295,
 43.926995,
 115.699814,
 134.279229,
 155.790452,
 57.925623,
 107.928811,
```

145.847379,
49.046956,
99.398009,
144.448633,
135.413856,
52.035179,
120.965743,
89.866276,
37.668301,
61.22601,
198.372395,
64.928382,
418.708552,
93.004993,
35.149586,
77.68208,
3.857526,
21.939663,
116.840296,
89.270217,
73.79505,
51.872839,
71.862892,
102.322217,
81.834855,
118.078691,
78.291018,
43.349855,
62.169881,
56.523205,
28.529607,
75.674458,
125.114374,
65.352913,
22.57178,
45.274225,
85.36908,
42.741719,
187.322927,
120.725053,
40.748915,
60.034162,
90.23792,
79.456485,
44.640292,
32.852443,
77.300194,

110.620647,
243.791743,
41.796339,
72.225265,
203.73459,
48.775723,
69.405188,
481.098624,
167.58371,
9.476999,
108.849621,
6.909942,
3.195174,
127.284427,
35.580815,
21.133748,
63.148529,
202.042635,
25.468493,
39.019229,
56.758411,
70.867401,
115.040024,
81.781591,
5.954334,
73.313918,
70.78591,
99.499595,
434.278564,
86.493424,
26.710398,
114.522237,
33.769336,
63.079003,
39.004588,
54.159392,
66.21806,
58.485967,
91.332849,
35.601665,
56.747978,
77.77477,
52.341226,
101.599427,
73.616808,
82.502566,
100.21391,

```
17.88953,
64.798873,
24.855701,
34.981698,
59.325589,
21.605568,
26.074908,
74.506246,
39.604363,
62.898336,
45.381501,
120.09361,
76.310119,
38.068736,
20.678787,
48.829437,
42.840582,
68.757242,
39.448066,
271.972889,
2.871739,
7.255718,
81.499621,
47.686442,
43.129703,
27.220157,
90.33681,
50.073575,
41.380094,
54.931334,
60.810723,
51.328145,
46.834704,
62.641286,
33.616115,
67.33767,
27.835436,
66.757869,
17.060695,
44.906918,
67.698004,
44.108427,
43.987061,
15.953444,
54.035265,
18.714197,
60.467984,
```

143.350376,
16.90444,
39.873791,
38.126095,
48.780039,
100.412364,
42.887121,
23.657284,
44.635452,
31.703608,
71.510596,
37.195046,
23.336875,
45.94834,
48.93337,
138.049577,
22.643776,
95.914473,
44.490453,
37.68056,
52.995628,
95.301296,
65.948959,
88.935165,
51.188633,
101.74155,
38.729418,
24.399642,
24.507987,
53.732892,
58.849256,
30.863434,
41.426678,
20.652943,
52.792678,
48.205606,
58.782359,
32.271938,
76.842247,
61.025639,
33.507289,
50.767332,
12.092241,
30.254021,
41.862983,
19.224754,
44.455166,
109.528572,
123.630332,
106.914973,
41.498631,
1.859364,
138.433168,

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

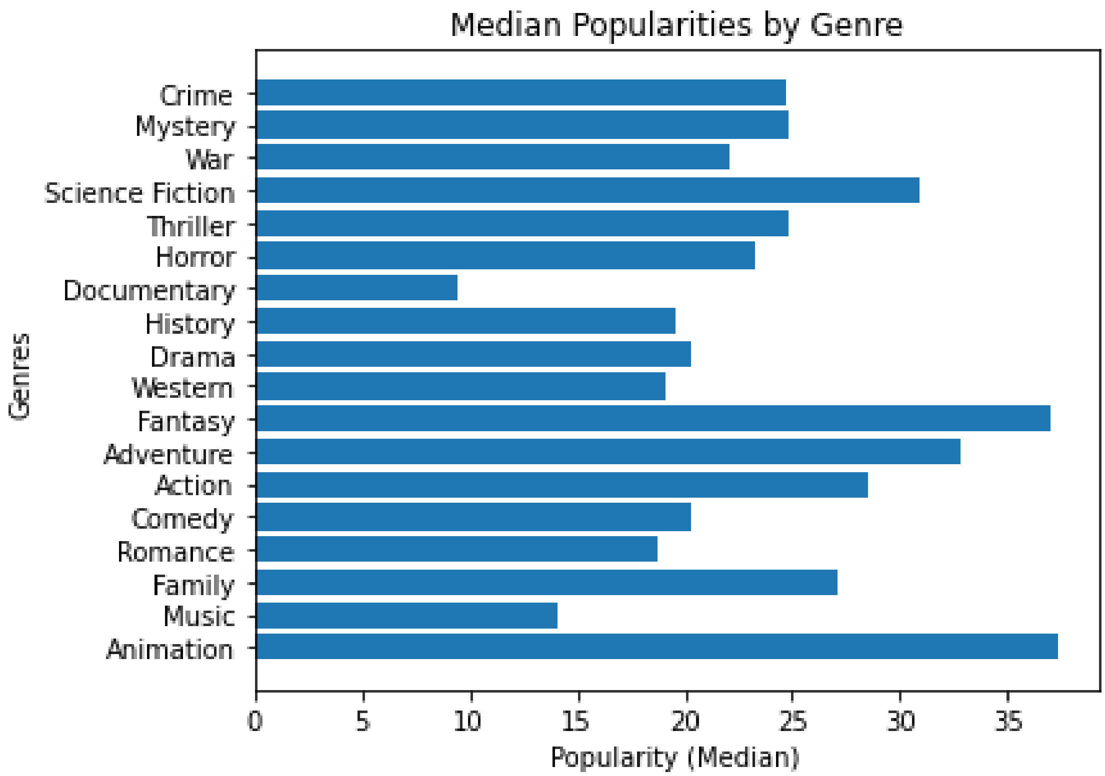
[14]:
```
['Animation',
 'Music',
 'Family',
 'Romance',
 'Comedy',
 'Action',
 'Adventure',
 'Fantasy',
 'Western',
 'Drama',
 'History',
 'Documentary',
 'Horror',
 'Thriller',
 'Science Fiction',
 'War',
 'Mystery',
 'Crime']
```

[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)
```

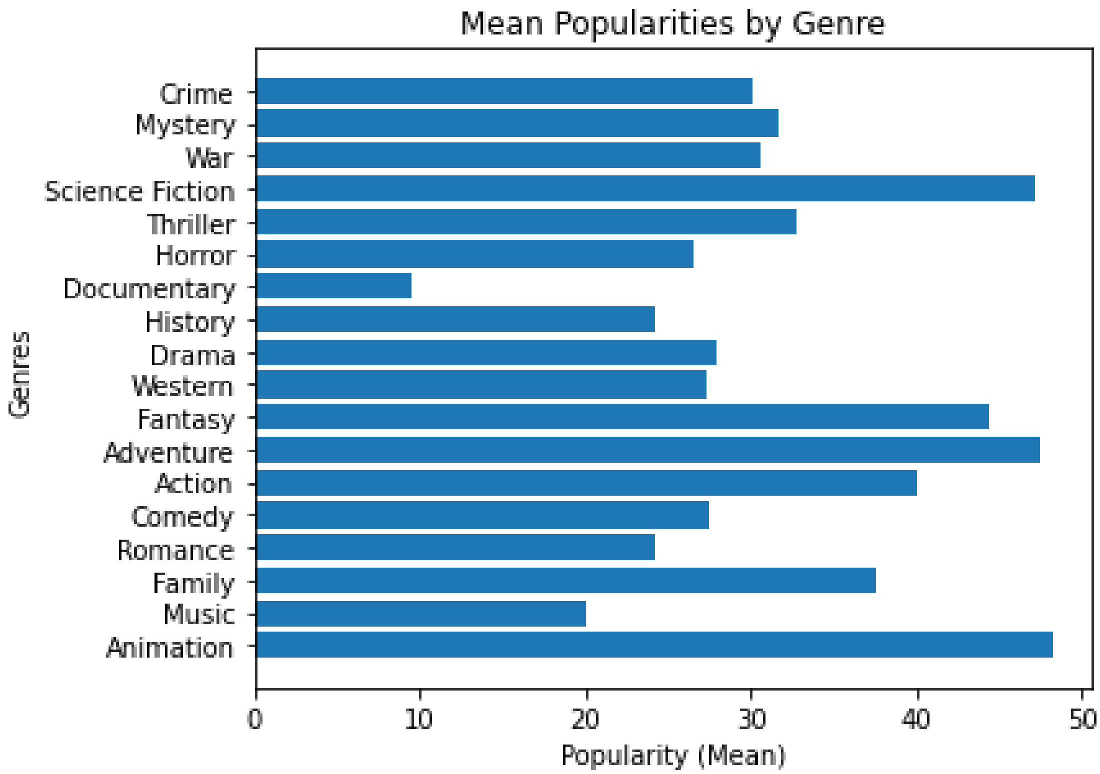
[17]:
```
# Plot data
pyplt.barh(y=genres,width=median_val_rounded)
pyplt.tight_layout()
pyplt.xlabel("Popularity (Median)")
pyplt.ylabel("Genres")
pyplt.title("Median Popularities by Genre")
```

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



[18]:
```
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.

-

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

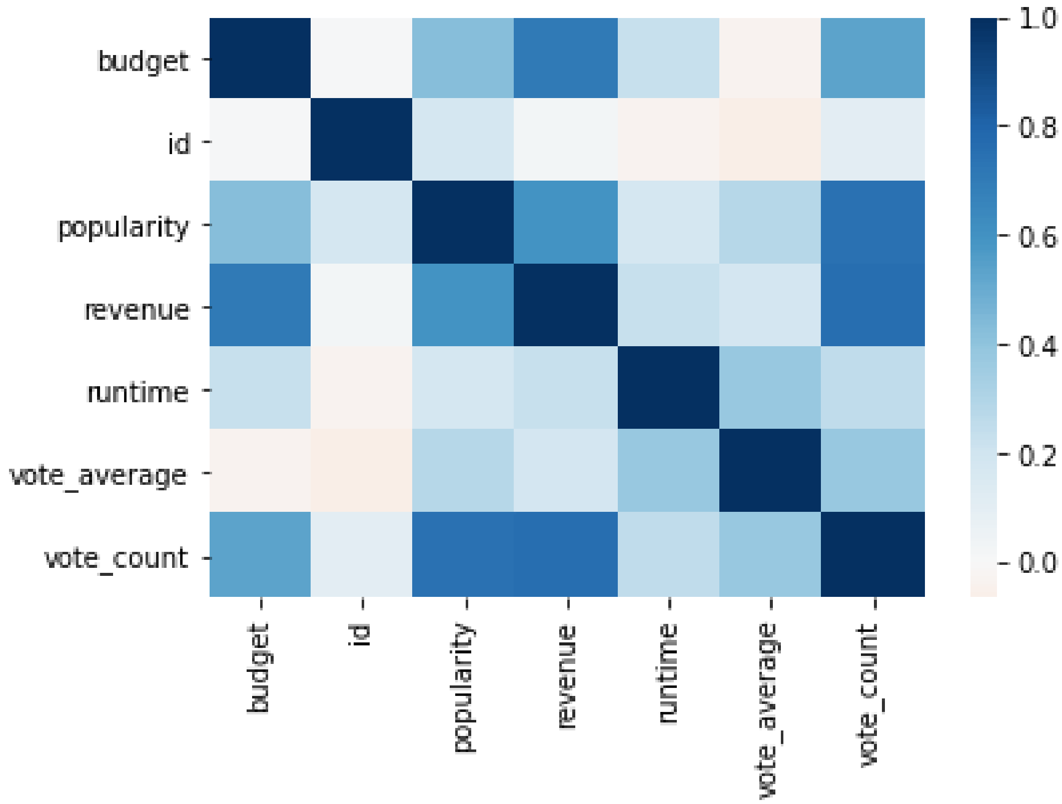
[19]:

	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.

0.1.2 Modeling and Question Answering

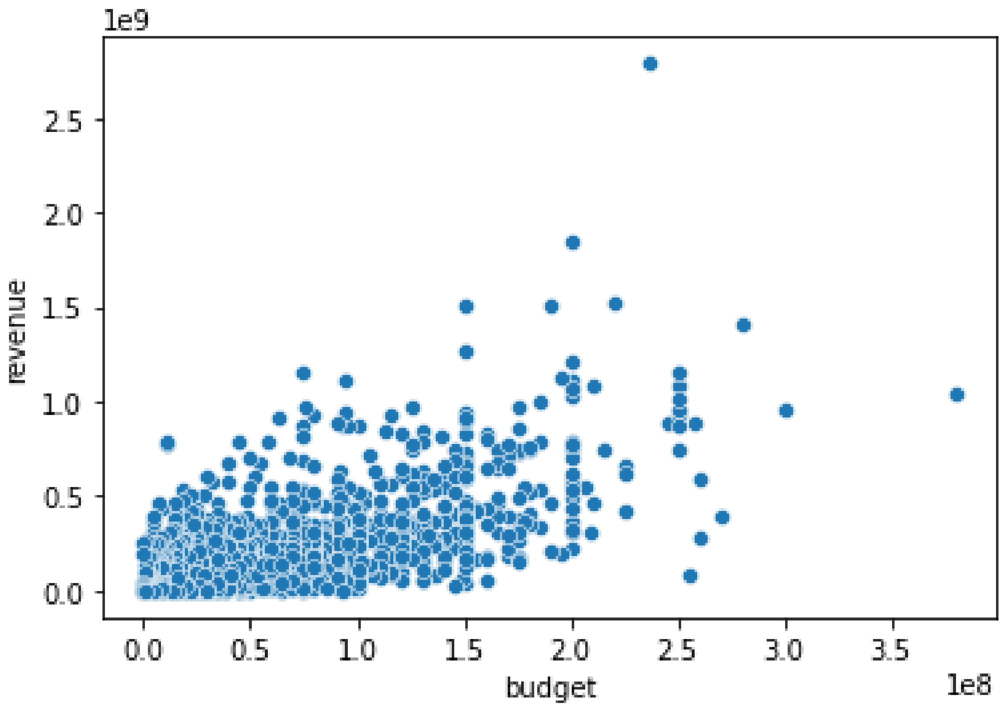
[22]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
```

-

[23]:
```python
# 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'>

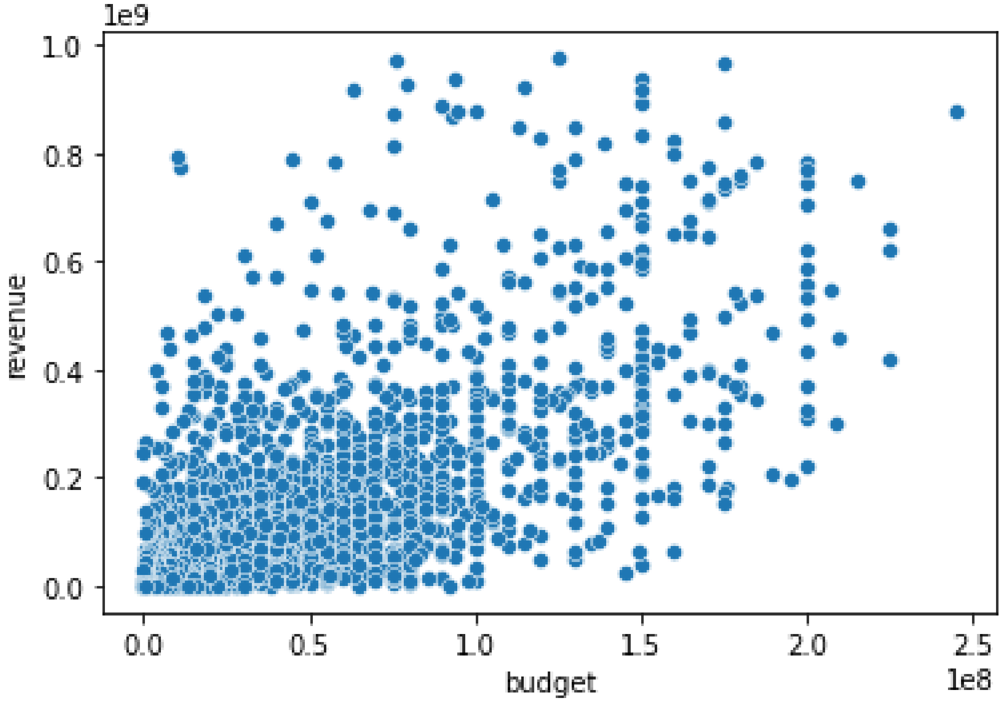


[24]:
```
# Remove the outliers according to the scatter plot.
data_without_outliers = data[(data["budget"] < 250000000) & (data["revenue"] <␣
↪1000000000)]
```

[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 = 2.565989921721056 * Budget +
12151647.736499479
```

[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)
```

```
Average MSE: 1.3211146352793626e+16
```

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.

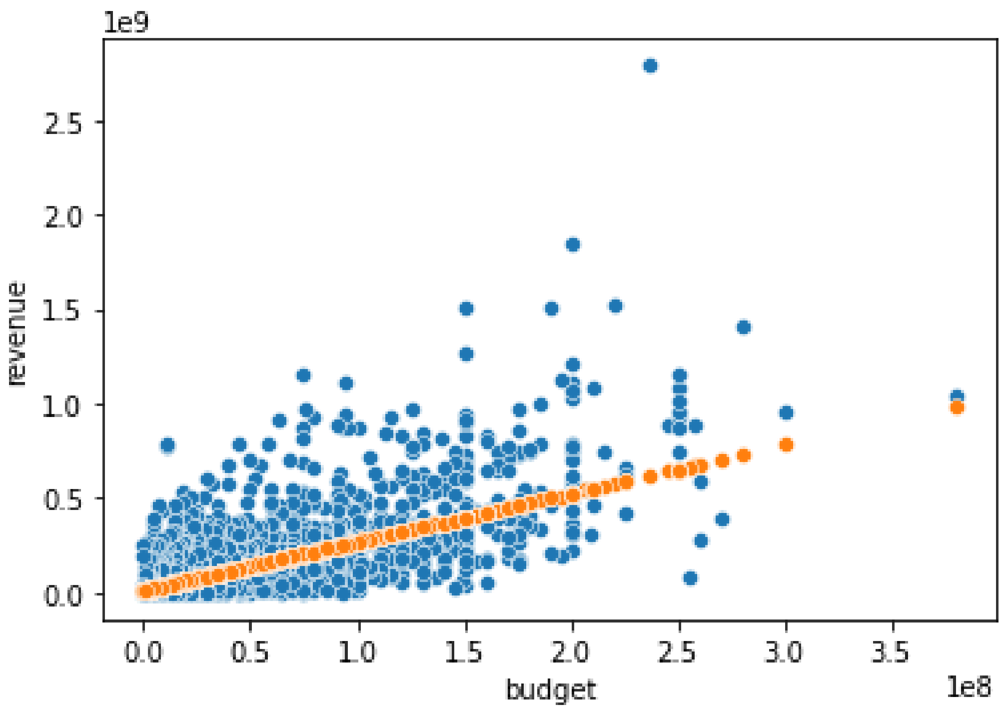
[28]:
```
# By the model, calculate the predicted values of revenue.
predictions = model_1.predict(data["budget"].to_numpy().reshape(-1, 1))
```

[29]:
```
# 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 = 2.5713786556233975 * Budget +
12124050.399081381
```

[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)
```

```
Average MSE: 1.2933494929720106e+16
```

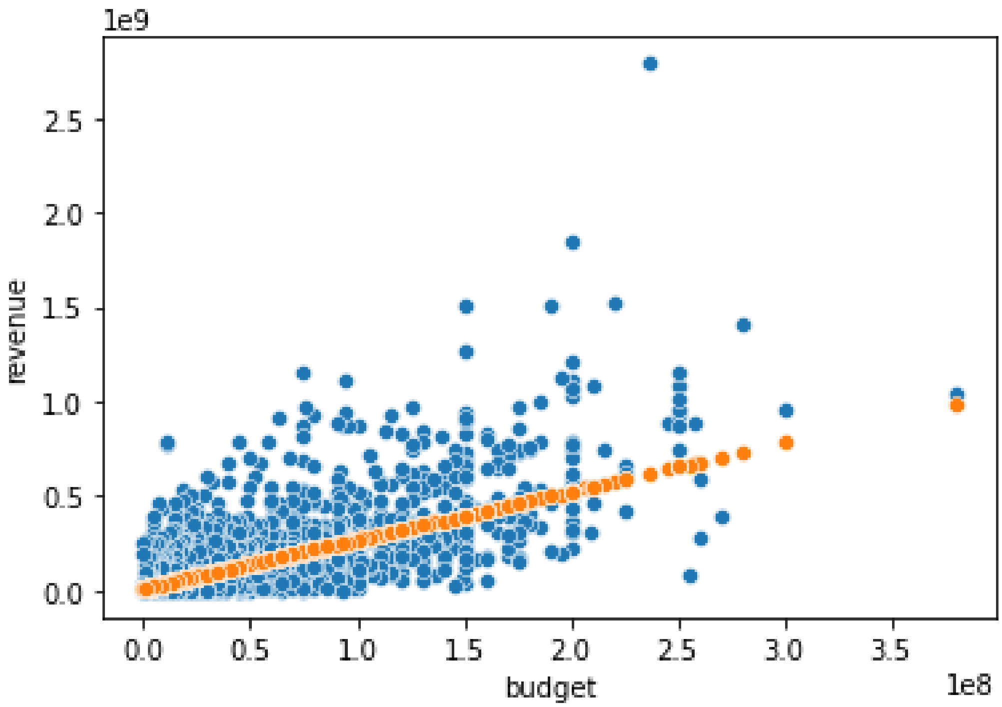
[32]:
```
# By the model, calculate the predicted values of revenue.
predictions = model_1.predict(data["budget"].to_numpy().reshape(-1, 1))
```

[33]:
```
# 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.

-

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
```

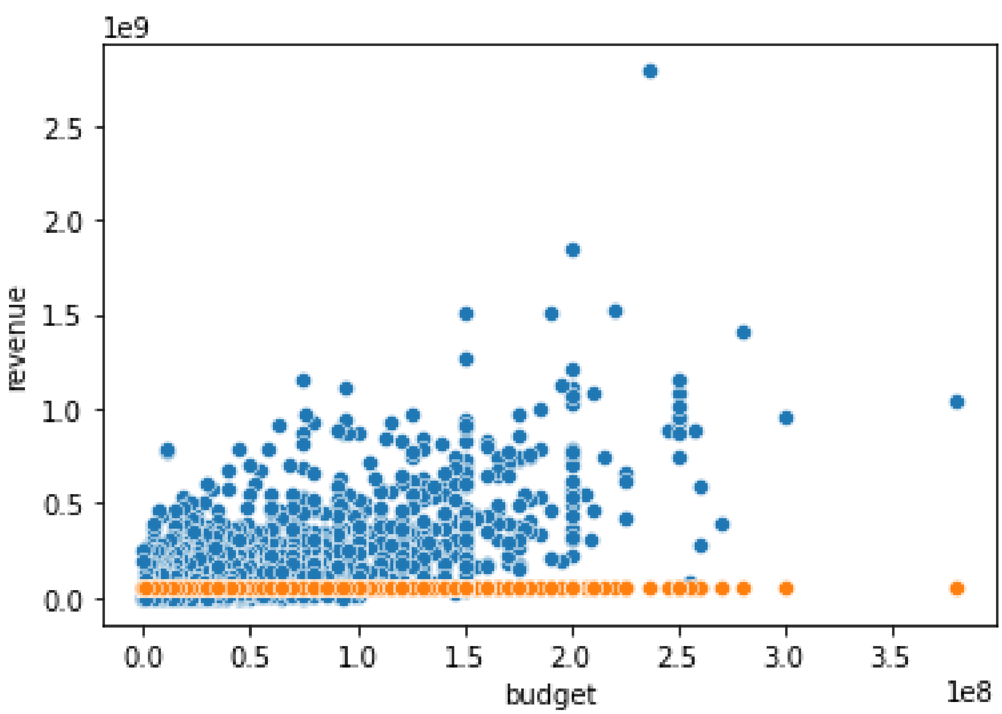
[35]:
```
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE, for basem1:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
```

```
Average MSE, for basem1: 3.084953711419822e+16
Average RMSE: 175.64036299836724 million
```

[36]:
```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"], 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.
    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_))
```

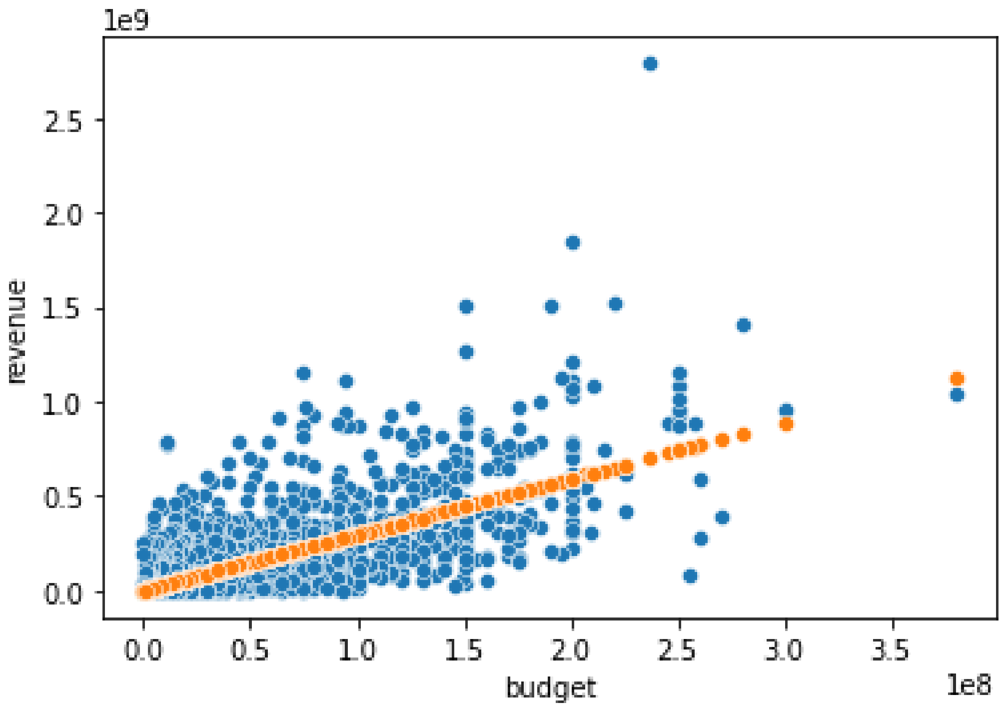
```python
# Calculate the average of each linear regression model's MSE in the loop.
MSE_average = np.mean(MSE)
# Print out the average.
print("Average MSE:", MSE_average)
print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million')
sns.scatterplot(data["budget"], data["revenue"])
sns.scatterplot(data["budget"], 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 = 2.959983869894621 * BudgetSquared +
1278862.4659905878
Average MSE: 1.7779793678443568e+16
Average RMSE: 133.34089274653732 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]:
```
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)
```

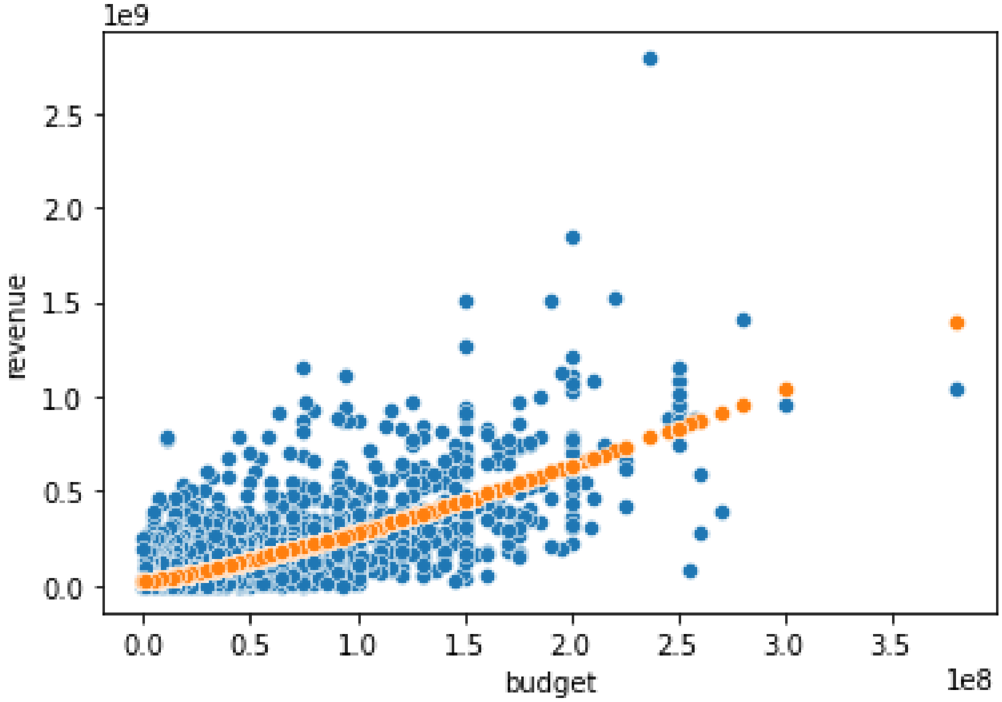
```
<ipython-input-38-bd3cf0b8d27f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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 = 0.02578263943398939 * BudgetSquared +
25075765.488036998
Average MSE: 1.7421227007868844e+16
Average RMSE: 131.98949582398154 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()
# data3.describe()
# data4.describe()
# pyplt.hist(data1.budget, bins=10)
# pyplt.hist(data2.budget, bins=10)
# pyplt.hist(data3.budget, bins=10)
```

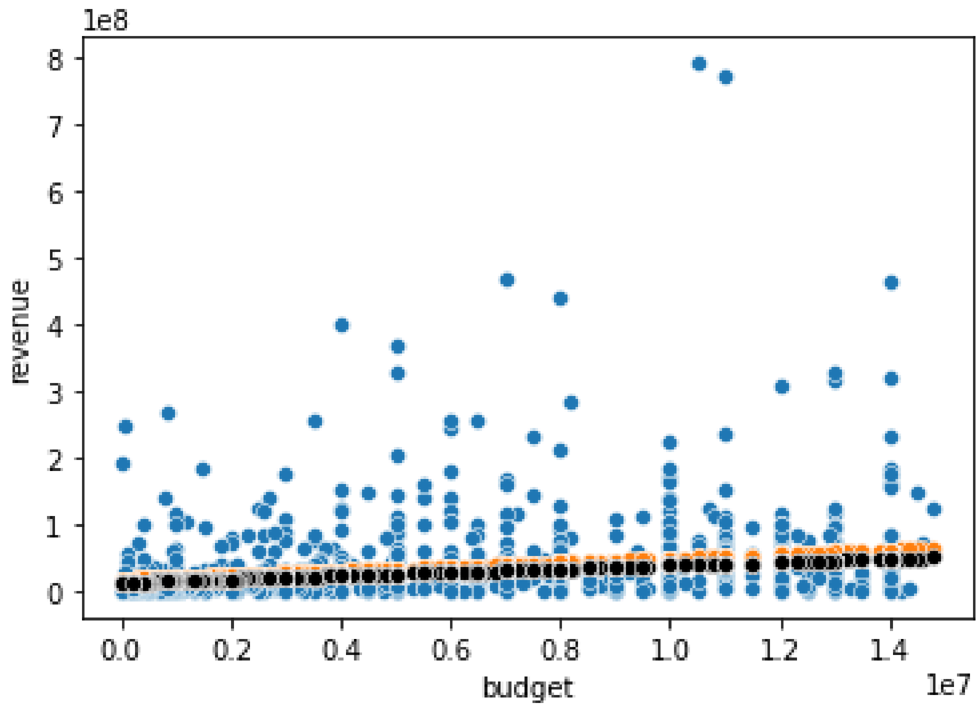
```
# pyplt.hist(data4.budget, bins=10)
```

[40]:
```
# 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
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 = 3.1709229013841105 * Budget +
16812924.068037845
Average MSE: 4170151027443897.0
Average RMSE: 64.57670653915308 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]:
```
# 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
```

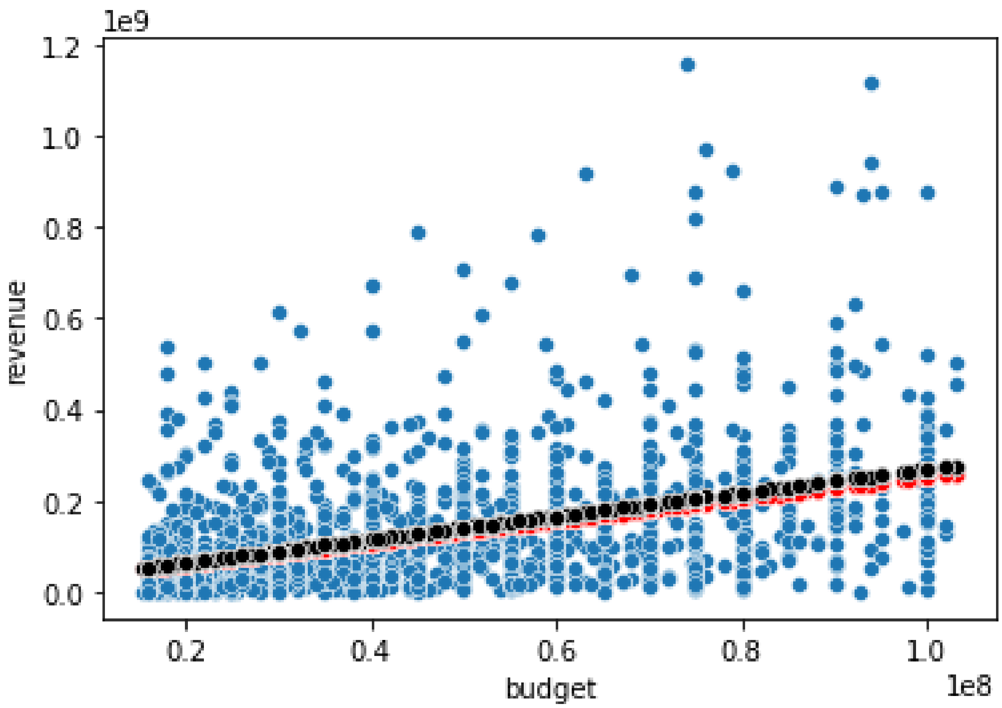
```
# Use the average of the value of slope (coefficient) as the slope of the fianl␣
↪model.
model_32.coef_ = np.array([np.mean(coef)])
# Use the average of the value of intercept as the intercept of the fianl model.
model_32.intercept_ = np.mean(intercept)
# Print the final linear regression model.
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 = 2.3831947433719782 * Budget +
12391098.004415099
Average MSE: 1.4650249976505202e+16
Average RMSE: 121.03821700812186 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.
```

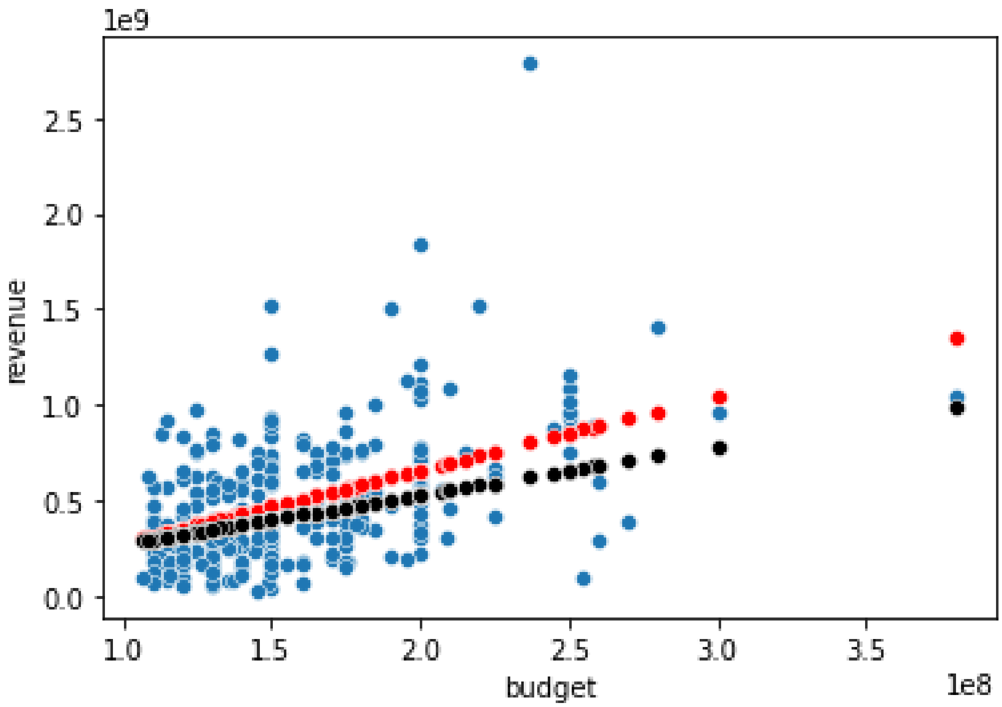
```
    train, test = train_test_split(data3)
    # Fit the dataset to the model.
    model_33 = model_33.fit(train["budget"].to_numpy().reshape(-1, 1),␣
↪train["revenue"].to_numpy())
    # Store the value of slope (coefficient) in each loop.
    coef.append(model_33.coef_[0])
    # Store the value of 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 = 3.8596046196373326 * Budget +
-113868041.15420528
Average MSE: 9.347146775003493e+16
Average RMSE: 305.7310382510008 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'>
```



•

[43]:
```
# My code starts here
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
import math

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(math.log(data["budget"].get(l)))
            y.append(math.log(data["revenue"].get(l)))
    allX[ug] = x
    allY[ug] = y
```

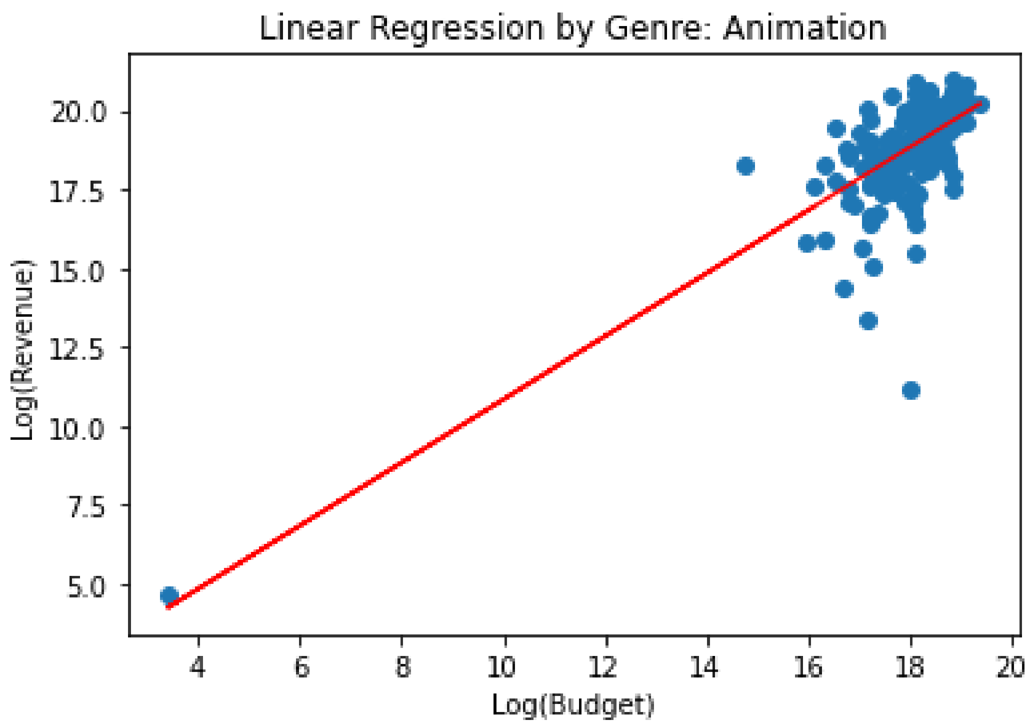
Here we sort all values of budgets (X) and revenue (Y) into two dictionaries with key values of the specific genre and the values of budget or revenue as lists depending on dictionary. In this process, repeats of the same movie does occur since most movies have more than a single genre. Additionally, 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)
```

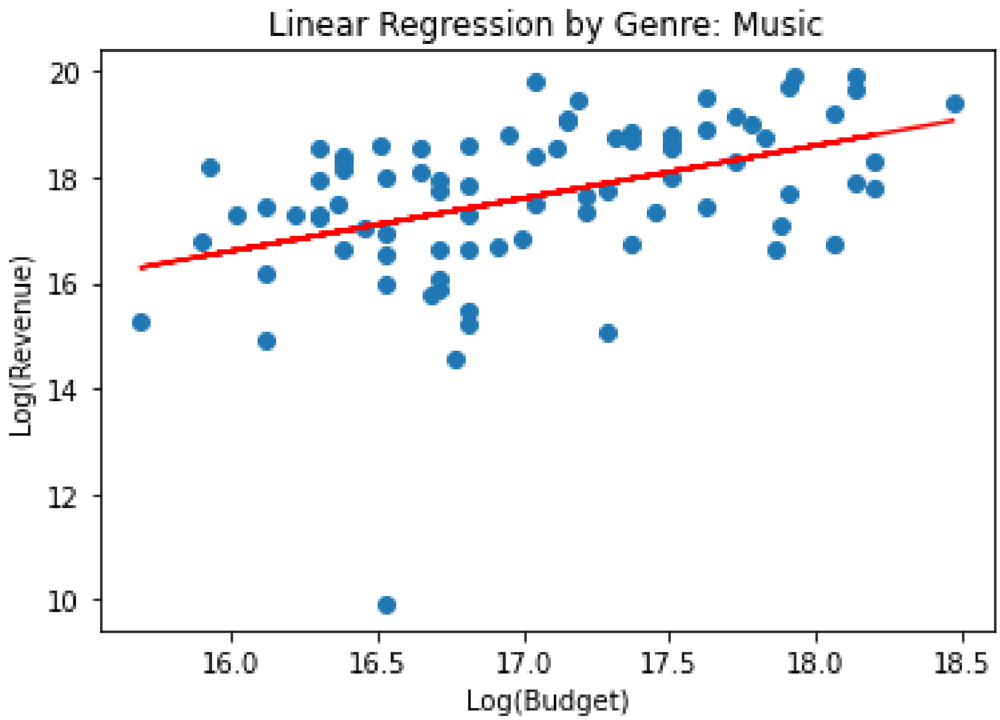
Here, we make sure that all genres have budgets and revenues by discarding the entire genre since an empty list of budget and revenues would result in an empty scatter plot.

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

for g in unique_genre:
    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("Log(Budget)")
        pyplt.ylabel("Log(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(data["budget"].to_numpy().reshape(-1, 1))
        MSE.append(np.mean((data["revenue"] - predictions) ** 2))
        m4coef_ = np.array([np.mean(coef)])
        m4intercept_ = np.mean(intercept)
```

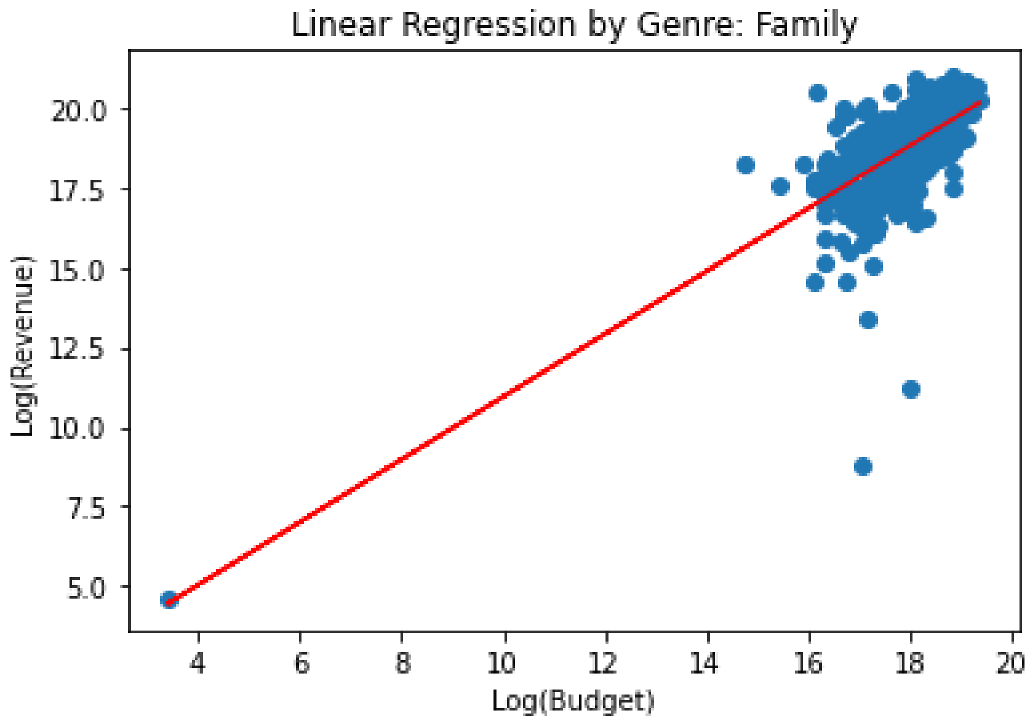
```
        print("The " + g + " movie linear regression model is: Revenue = " +␣
↪str(m4.coef_[0]) + " * Budget + " + str(m4.intercept_))

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


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")
```



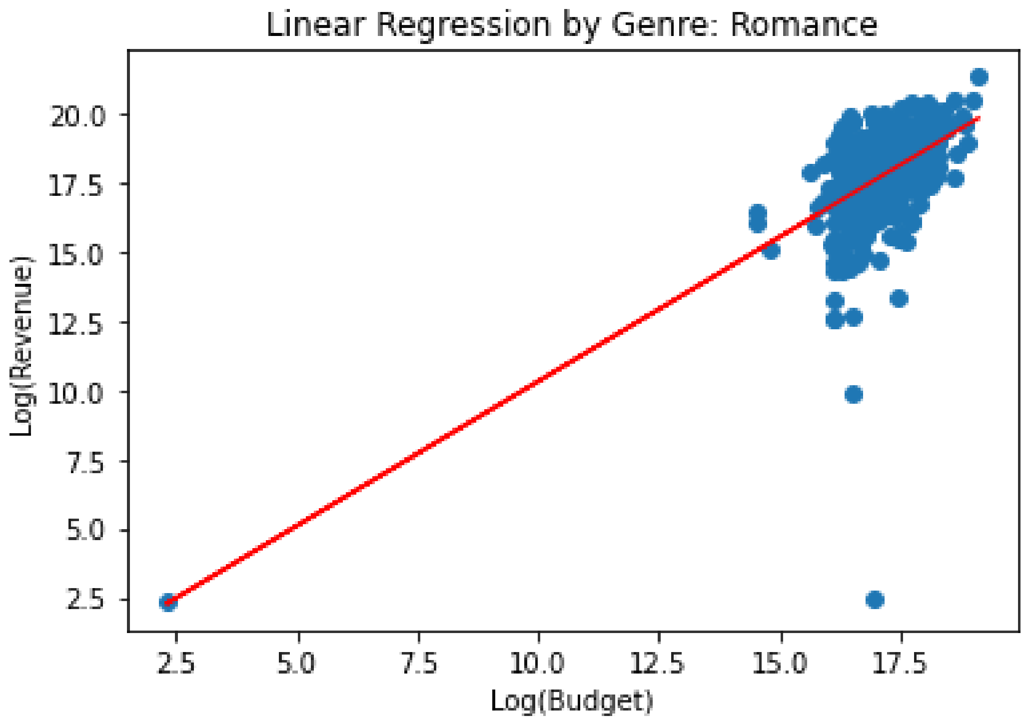

```
The Animation movie linear regression model is: Revenue = 0.9996253746508537 *
Budget + 0.8440999753806153
Average MSE, for model #4: 3.195472169694911e+16
Average RMSE: 178.75883669611724 million
```



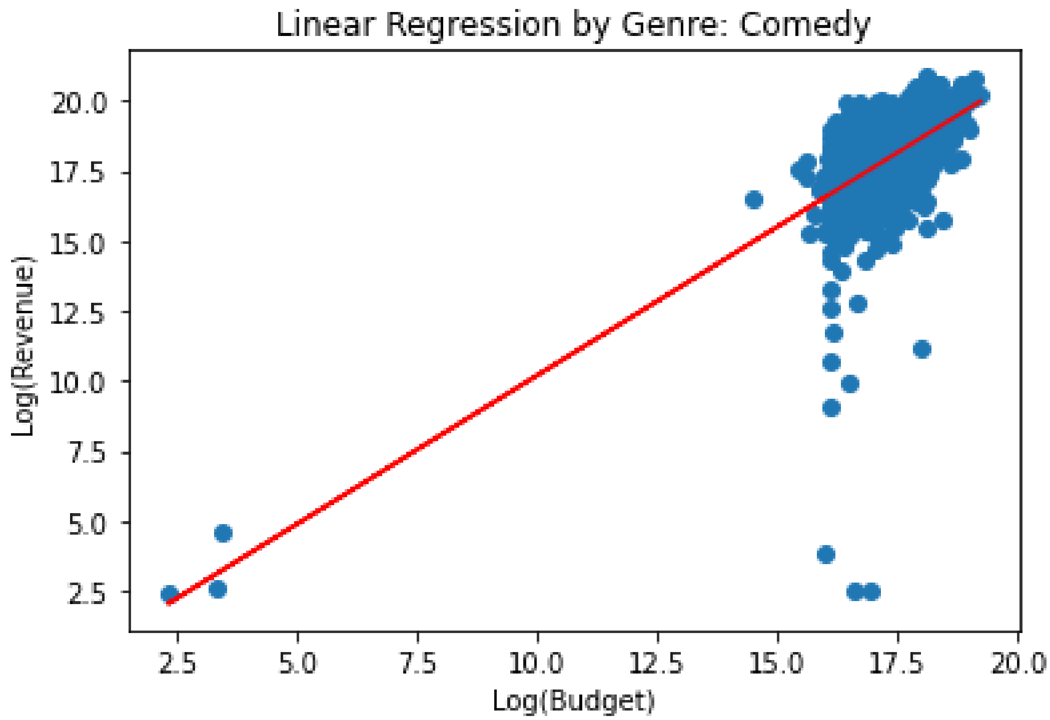

```
The Music movie linear regression model is: Revenue = 0.999738134287128 * Budget
+ 0.6080134153163286
Average MSE, for model #4: 3.195390471400805e+16
Average RMSE: 178.756551527512 million
```



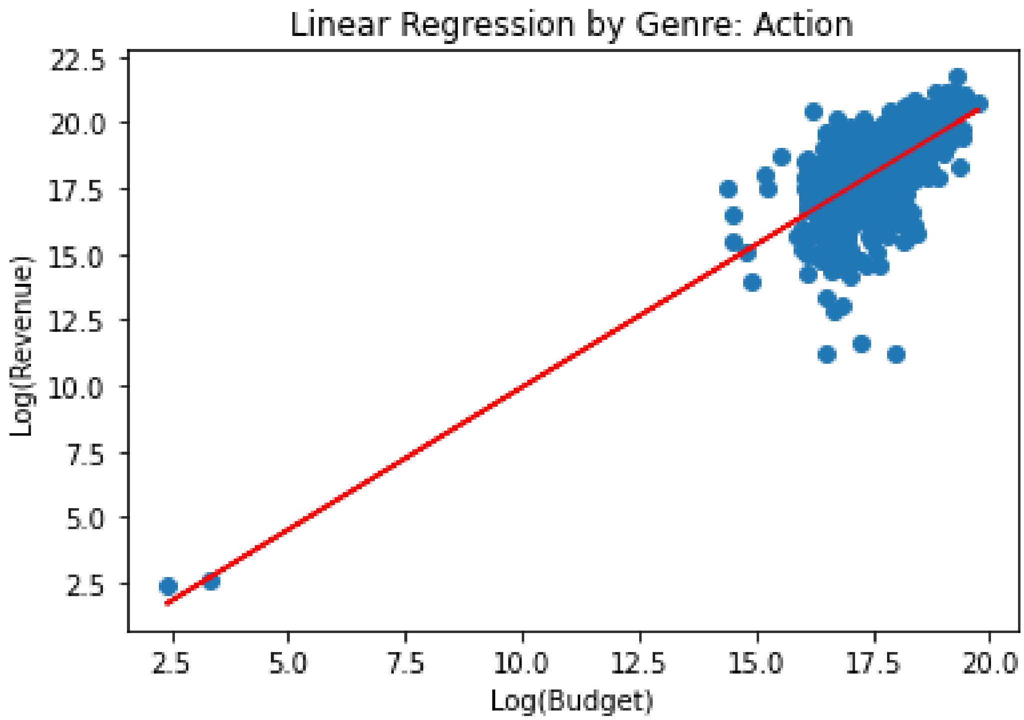

```
The Family movie linear regression model is: Revenue = 0.9822522774321161 *
Budget + 1.1175693113031606
Average MSE, for model #4: 3.203846693194387e+16
Average RMSE: 178.9929242510549 million
```



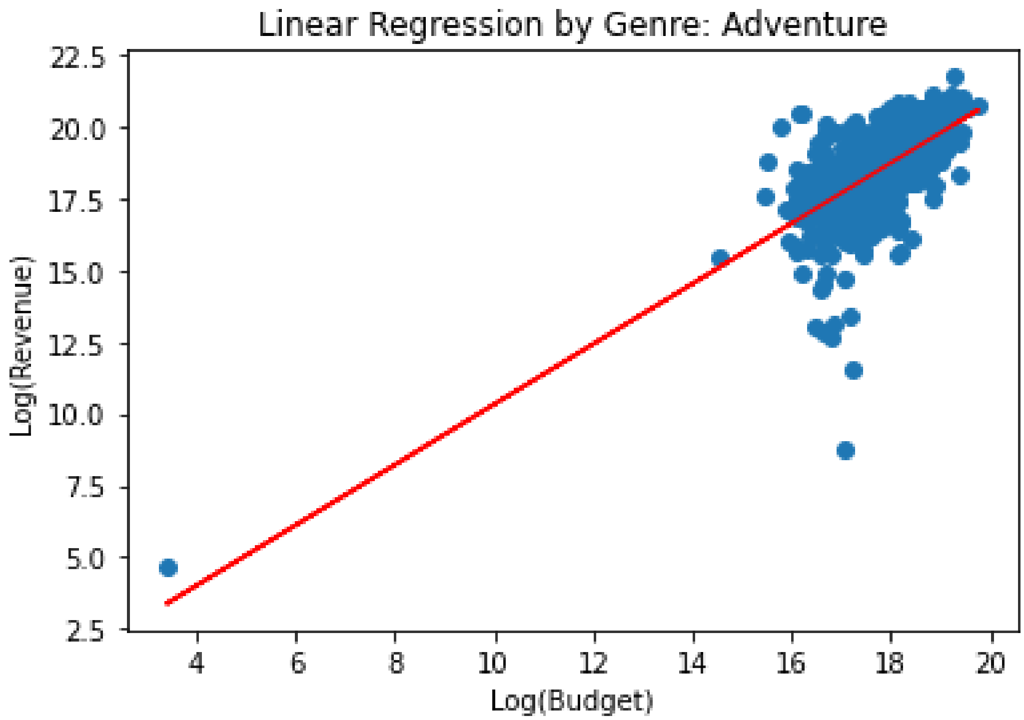

```
The Romance movie linear regression model is: Revenue = 1.0444788139833783 *
Budget + -0.13285509357041647
Average MSE, for model #4: 3.185687955939429e+16
Average RMSE: 178.4849561150583 million
```



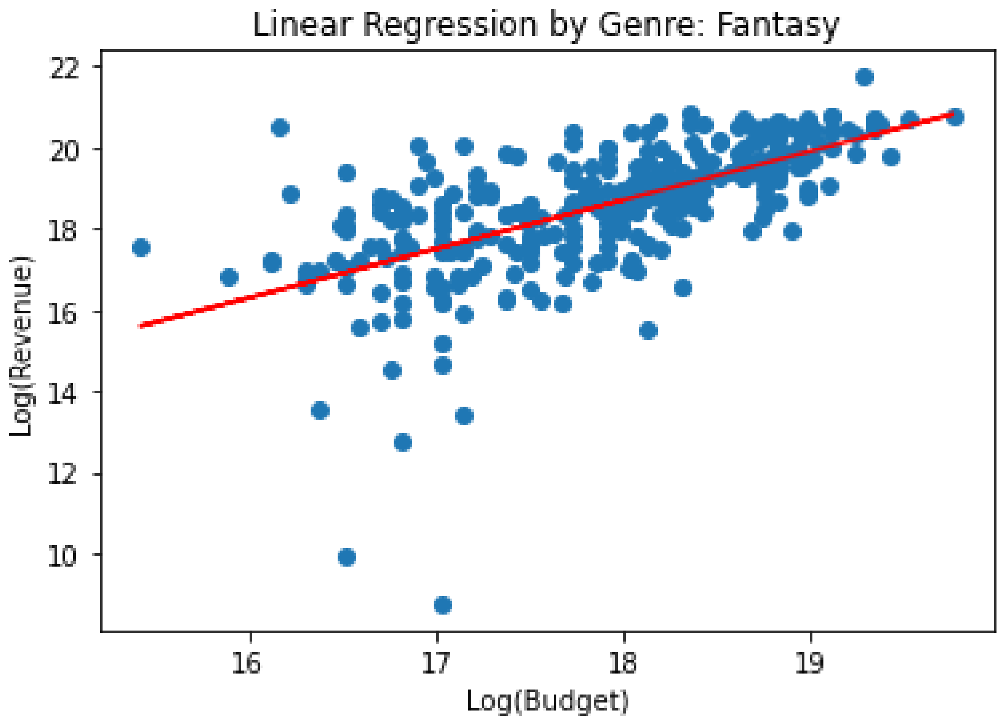

```
The Comedy movie linear regression model is: Revenue = 1.0603135683664549 *
Budget + -0.4112974517280641
Average MSE, for model #4: 3.17032607235656e+16
Average RMSE: 178.05409493624572 million
```



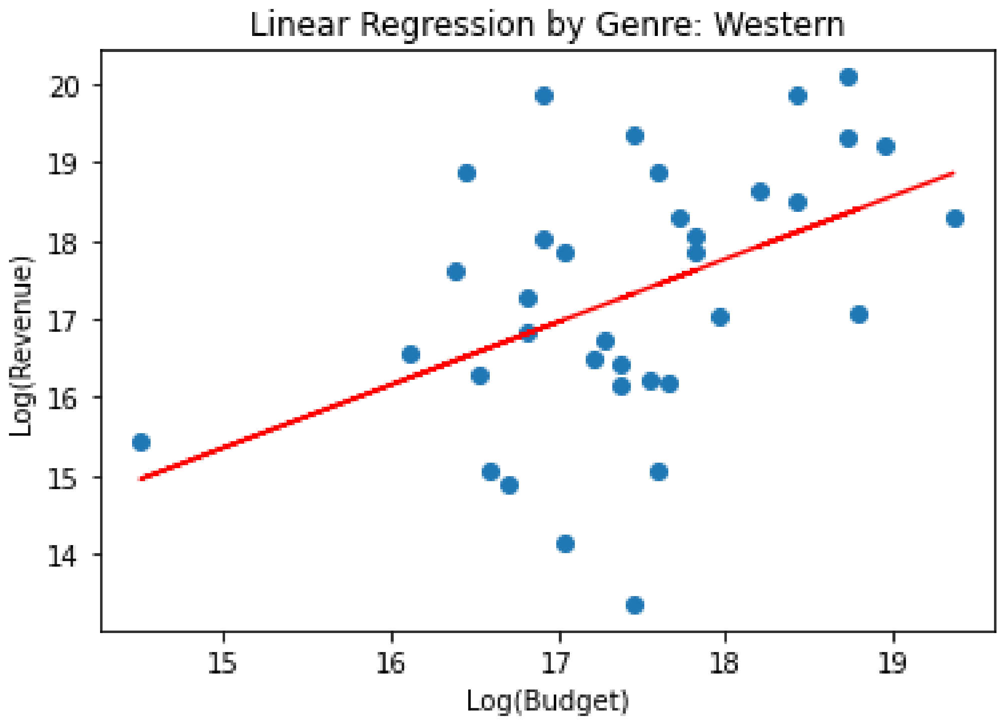

```
The Action movie linear regression model is: Revenue = 1.0833032120928778 *
Budget + -0.9009172232818798
Average MSE, for model #4: 3.1547353724129164e+16
Average RMSE: 177.61574739906695 million
```



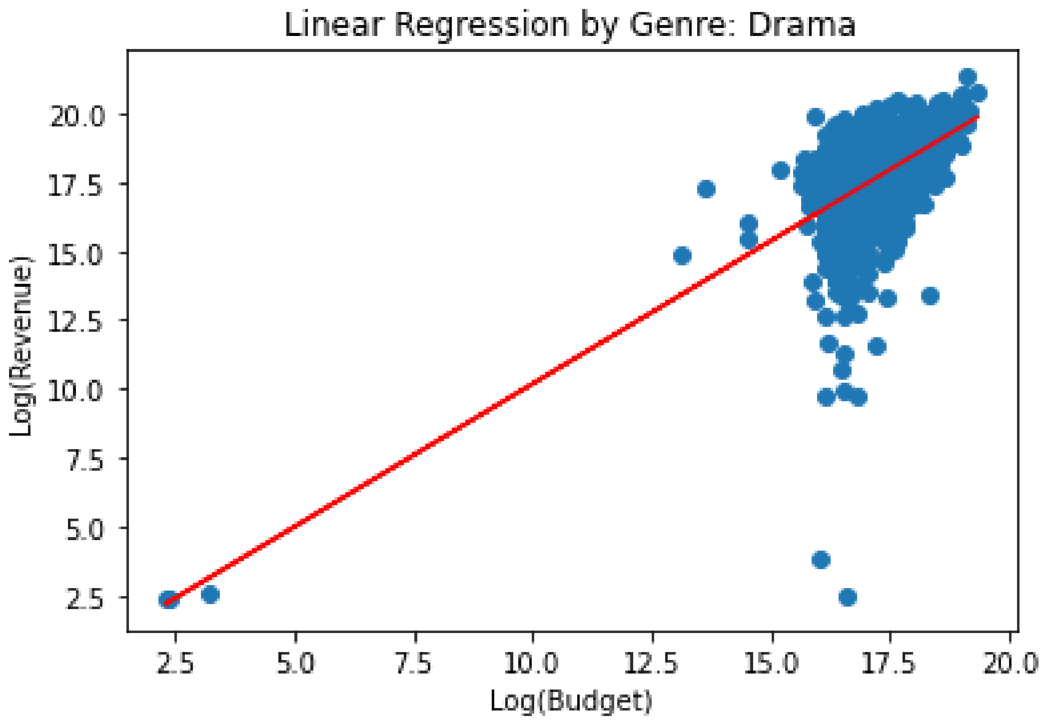

```
The Adventure movie linear regression model is: Revenue = 1.052670509615493 *
Budget + -0.2041529481709965
Average MSE, for model #4: 3.1497210697909724e+16
Average RMSE: 177.47453535059535 million

Average MSE, for model #4: 3.1497210697909724e+16
Average RMSE: 177.47453535059535 million
```



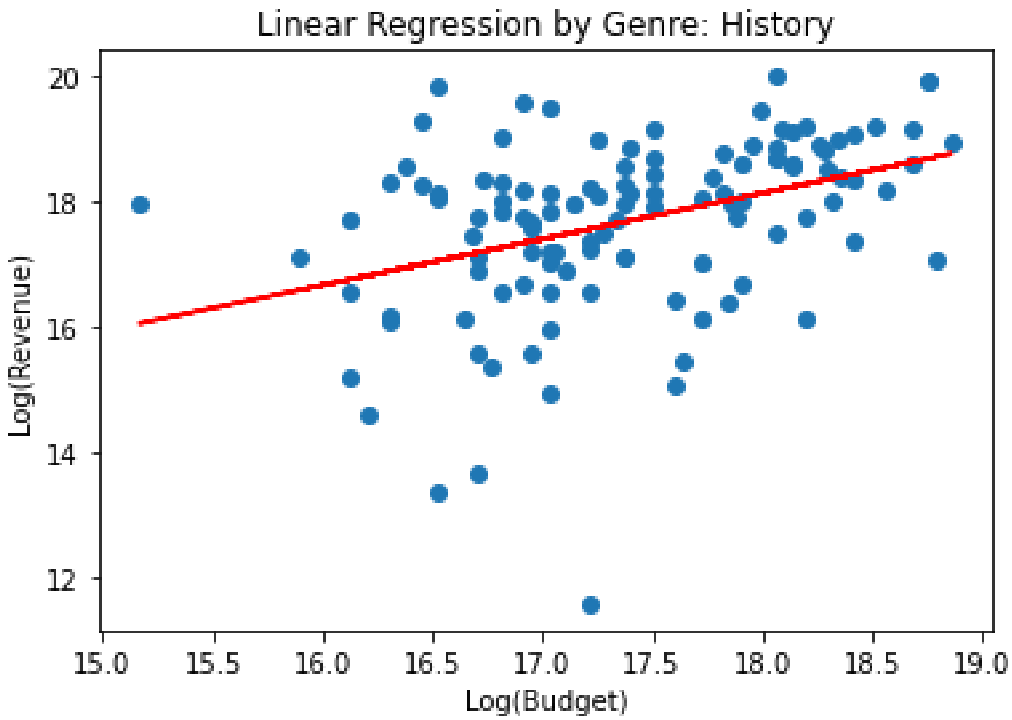

```
The Fantasy movie linear regression model is: Revenue = 1.2045125377912622 *
Budget + -2.974201891438412
Average MSE, for model #4: 3.120252807395184e+16
Average RMSE: 176.64237338179038 million
```



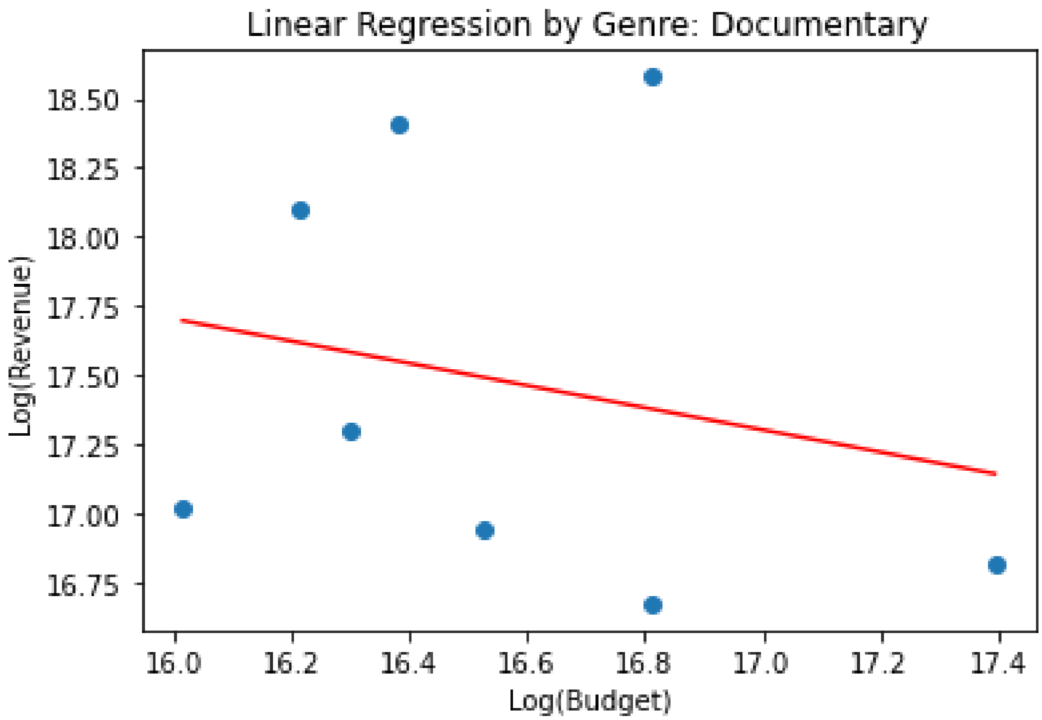

```
The Western movie linear regression model is: Revenue = 0.8050776197153847 *
Budget + 3.2734492825158608
Average MSE, for model #4: 3.1614800209055576e+16
Average RMSE: 177.80551231347013 million
```



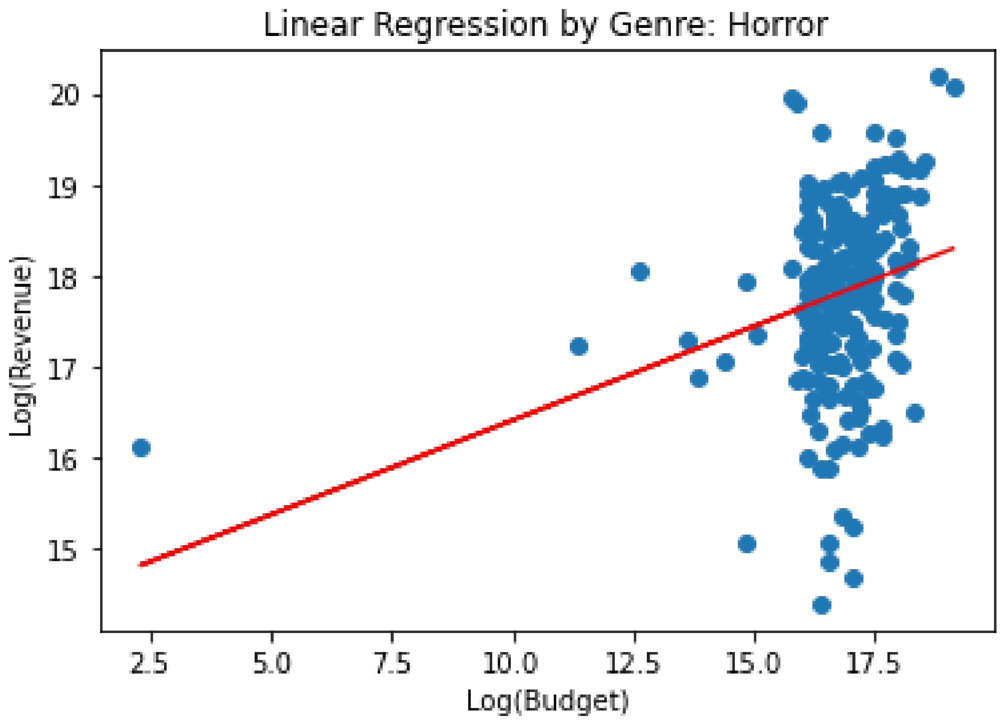

```
The Drama movie linear regression model is: Revenue = 1.0374826098906231 *
Budget + -0.19515802100759316
Average MSE, for model #4: 3.159445797426479e+16
Average RMSE: 177.74829949753328 million
```



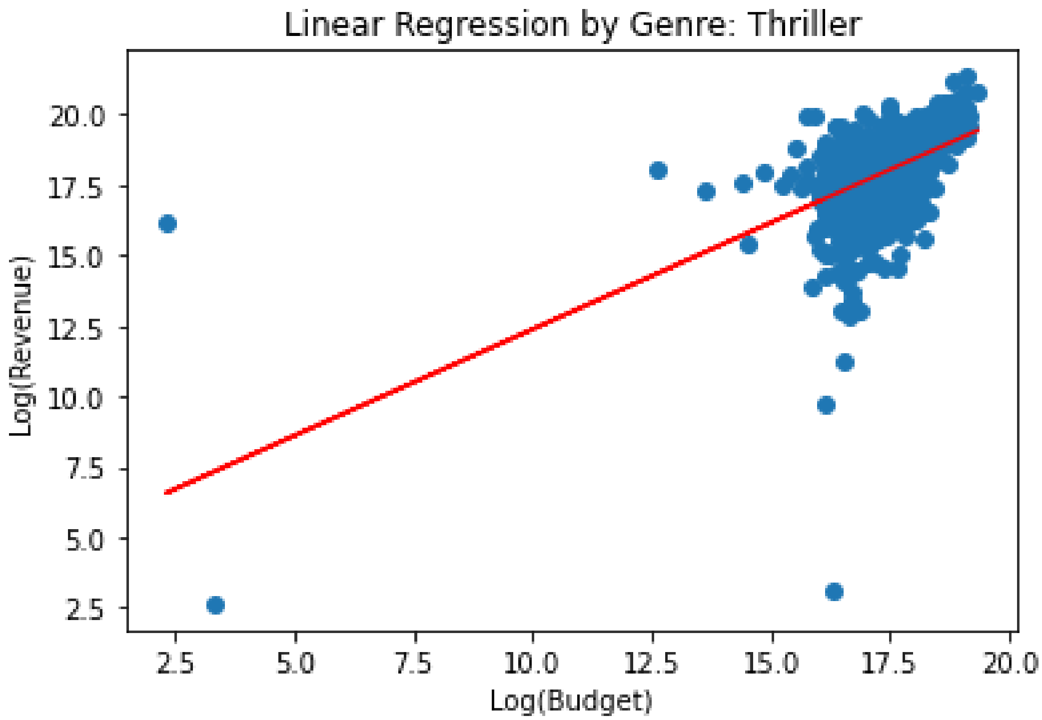

```
The History movie linear regression model is: Revenue = 0.7306690597077001 *
Budget + 4.980669121426153
Average MSE, for model #4: 3.2005680266624012e+16
Average RMSE: 178.9013143233554 million
```



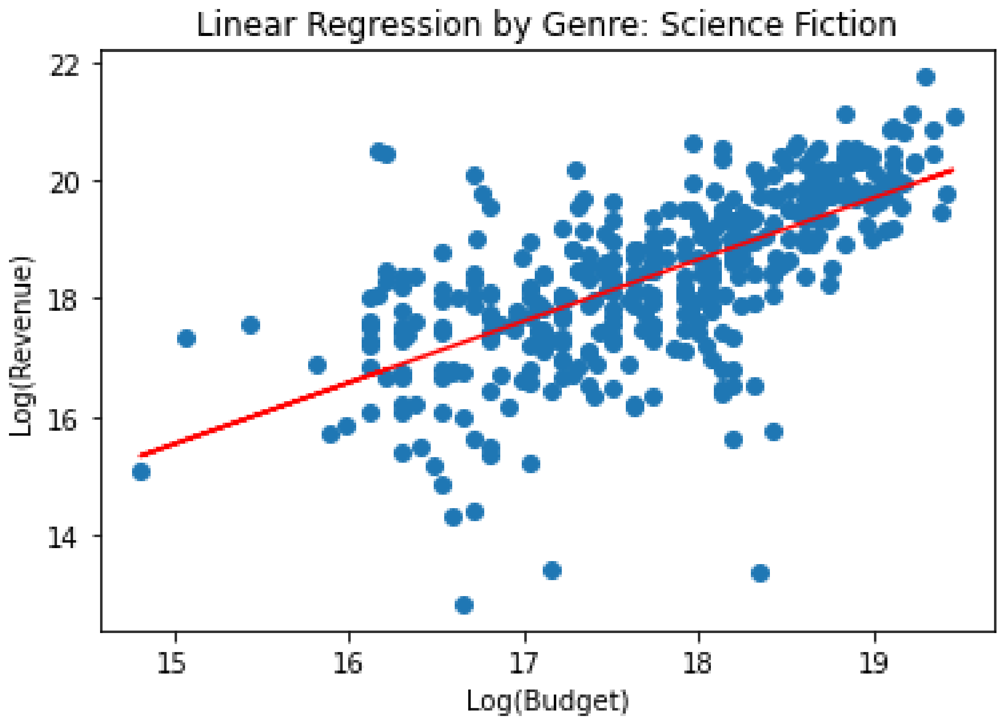

```
The Documentary movie linear regression model is: Revenue = -0.40035544983285326
* Budget + 24.10859408985679
Average MSE, for model #4: 3.429180221552374e+16
Average RMSE: 185.18045851418486 million
```



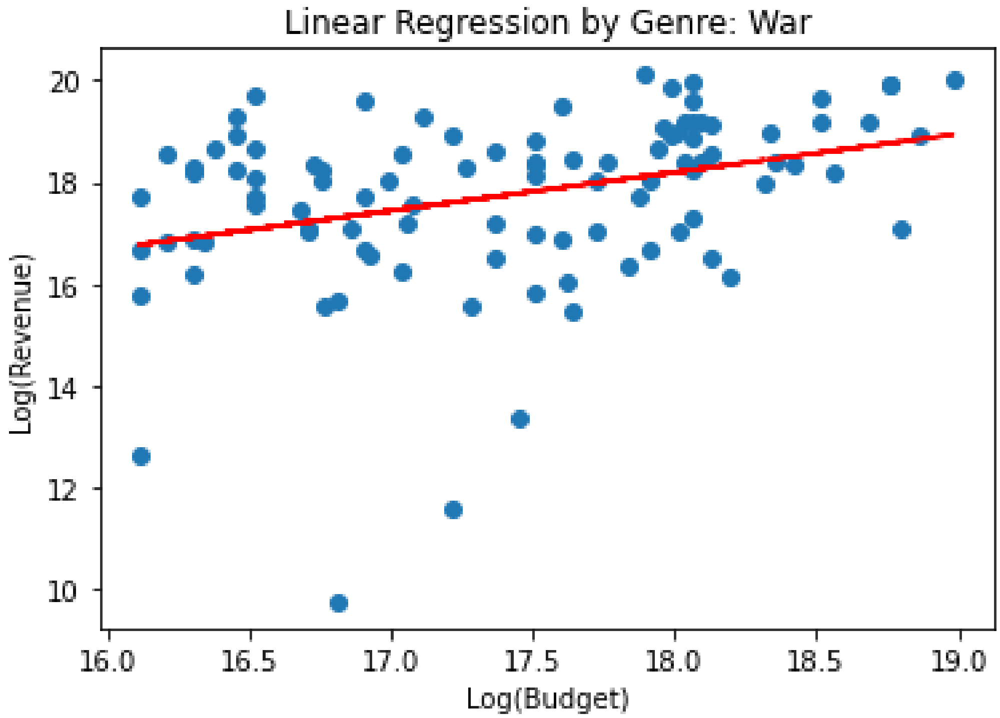

```
The Horror movie linear regression model is: Revenue = 0.2069959942669641 *
Budget + 14.340269563947485
Average MSE, for model #4: 3.5173045578801536e+16
Average RMSE: 187.5447828621248 million
```



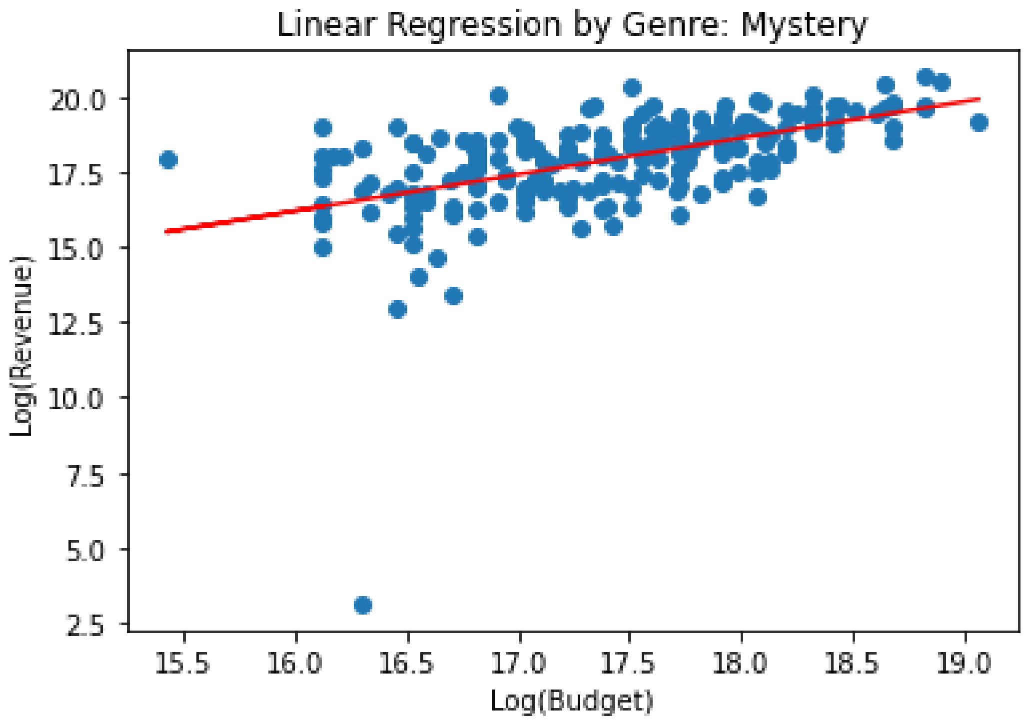

```
The Thriller movie linear regression model is: Revenue = 0.7550352027182395 *
Budget + 4.82581117874559950
Average MSE, for model #4: 3.52120320164082e+16
Average RMSE: 187.64869308473268 million
```



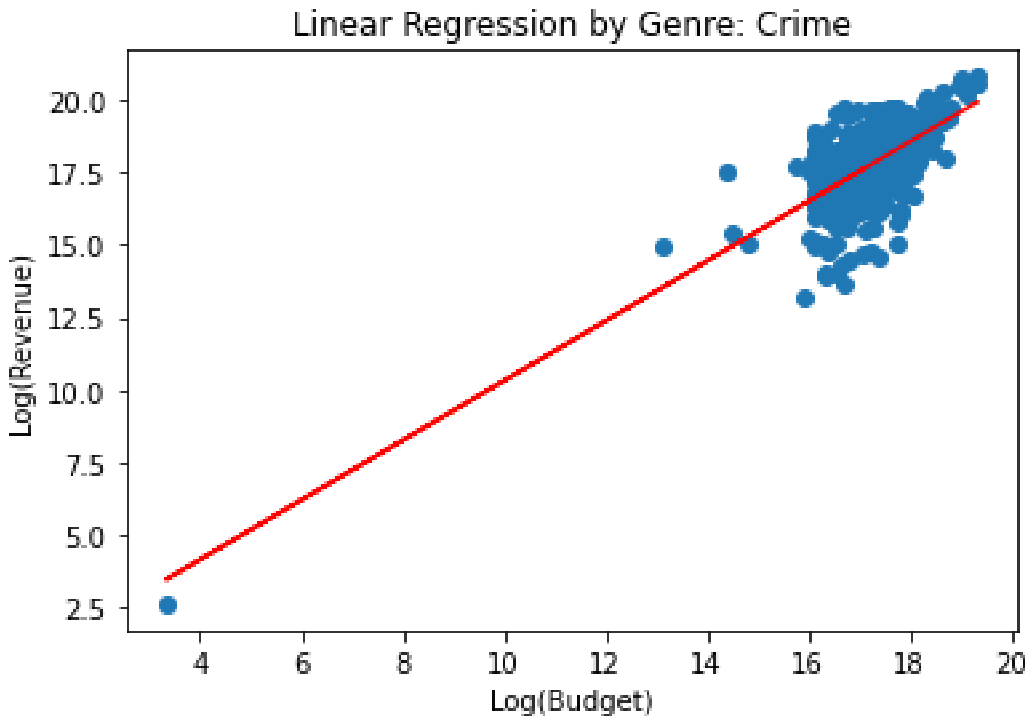

```
The Science Fiction movie linear regression model is: Revenue =
1.0416826093657396 * Budget + -0.09309070378832018
Average MSE, for model #4: 3.495467962669365e+16
Average RMSE: 186.96170631092787 million
```




```
The War movie linear regression model is: Revenue = 0.7533500753430058 * Budget
+ 4.632812389028199
Average MSE, for model #4: 3.50041567095266e+16
Average RMSE: 187.09397828237712 million
```




```
The Mystery movie linear regression model is: Revenue = 1.2132649141311718 *
Budget + -3.2207001264247346
Average MSE, for model #4: 3.4652521933290148e+16
Average RMSE: 186.15187867247042 million
```



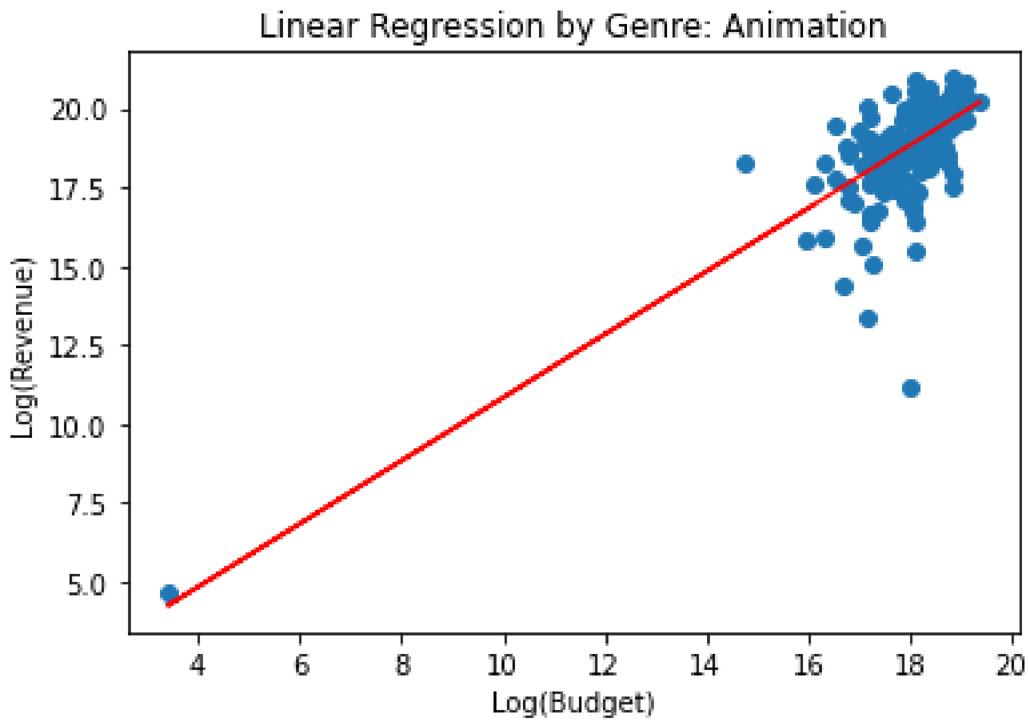

```
The Crime movie linear regression model is: Revenue = 1.0281119104877938 *
Budget + 0.05093834362745042
Average MSE, for model #4: 3.4479875722225756e+16
Average RMSE: 185.687575573127 million

Average MSE across all genres is 3.467969939289863e+16
Average RMSE across all genres is 191.2765974690189 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, average MSE and average RMSE of each genre are about 1.86e+16 and 140 million respectively. 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.

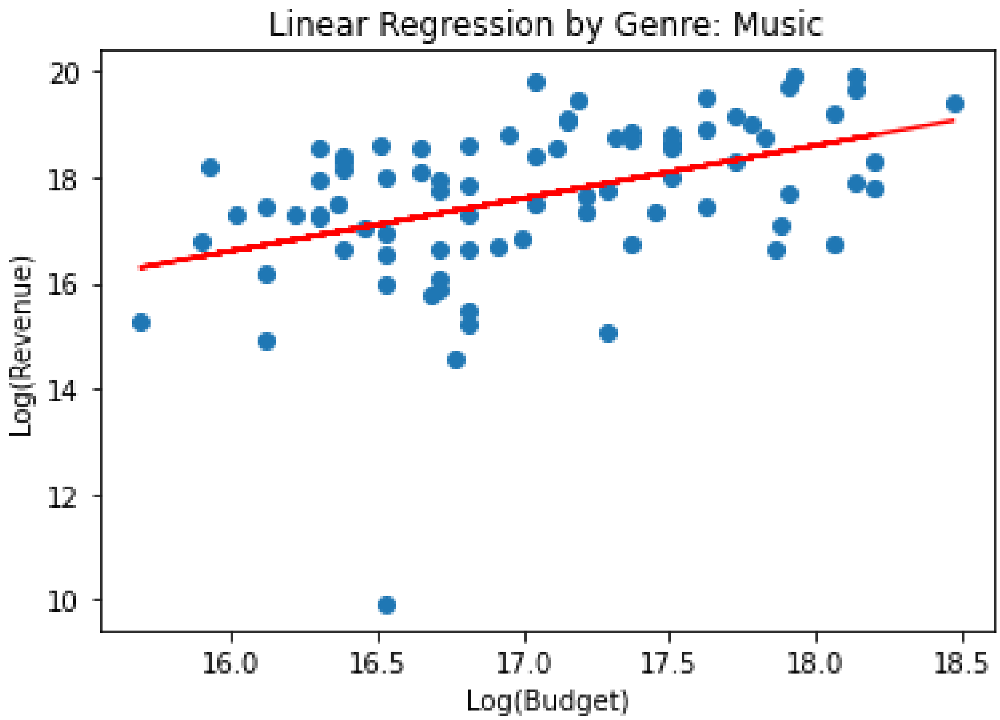
[46]:

```
coef = []
intercept = []
MSE = []
sum_MSE = 0
sum_RMSE = 0

for g in unique_genre:
    # Gets all coordinates for x and y
    if(allX.get(g) and allY.get(g)):
        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("Log(Budget)")
        pyplt.ylabel("Log(Revenue)")
        title = "Linear Regression by Genre: " + g
        pyplt.title(title)
        pyplt.show()

        # Applies cross-validation by spliting data
        kf = KFold(n_splits = 10, shuffle = True)
        for train_index, test_index in kf.split(data):
            train, test = train_test_split(data)
            m4 = m4.fit(train["budget"].to_numpy().reshape(-1, 1),␣
 ↪train["revenue"].to_numpy())
            coef.append(m4.coef_[0])
            intercept.append(m4.intercept_)
            predictions = m4.predict(test["budget"].to_numpy().reshape(-1, 1))
            MSE.append(np.mean((test["revenue"] - 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_average)
    print("Average RMSE:", MSE_average ** 0.5 / 1000000, 'million\n')
    sum_RMSE += MSE_average ** 0.5 / 1000000
    sum_MSE += MSE_average

print("Average MSE across all genres is " +str(sum_MSE/len(allX)))
```

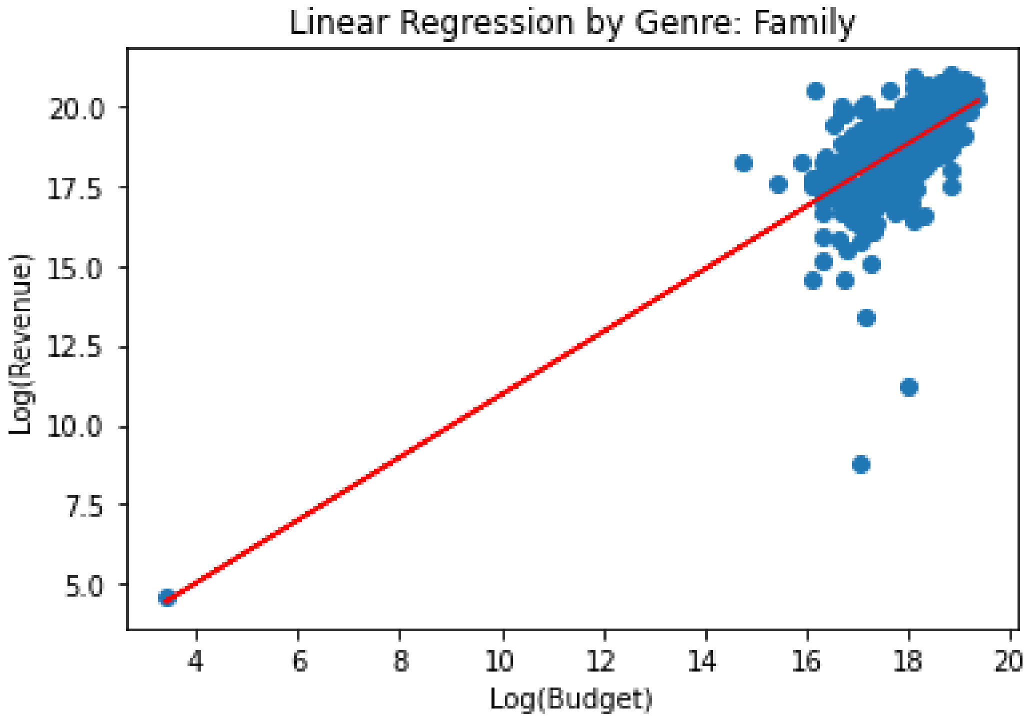
```
print("Average RMSE across all genres is " +str(sum_RMSE/len(allX)) + "␣
→million")
```



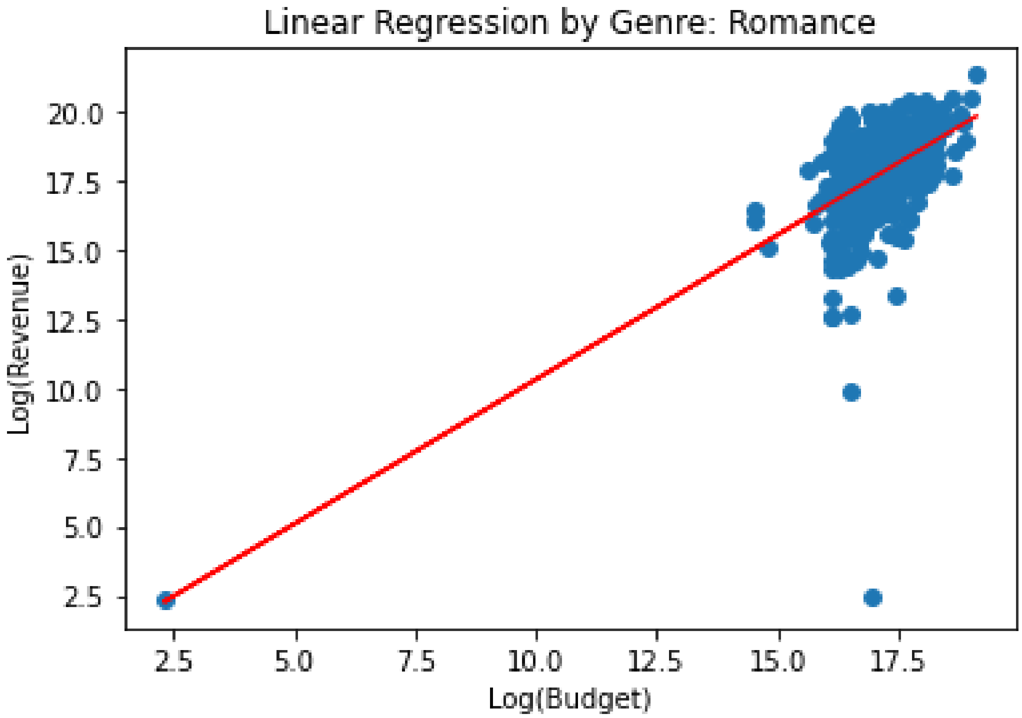

```
The Animation movie linear regression model is: Revenue = 2.8637635365566276 *
Budget + 3098560.186215684
Average MSE, for model #4: 1.6993226590611046e+16
Average RMSE: 130.35807067692835 million
```



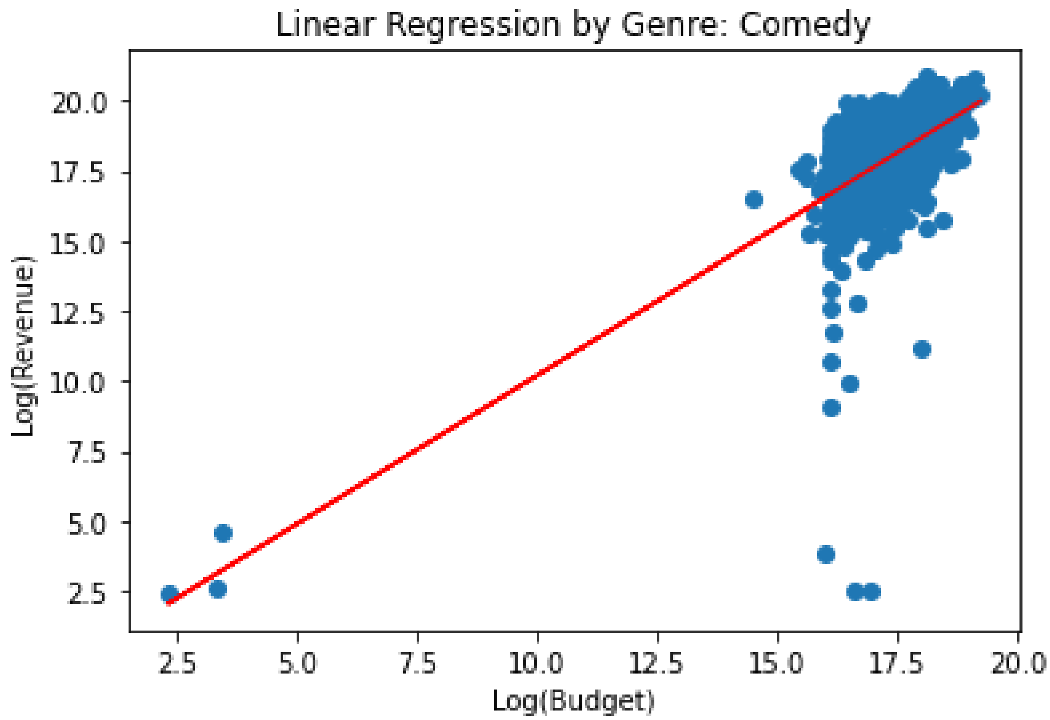

```
The Music movie linear regression model is: Revenue = 3.0210554137341497 *
Budget + -799255.1879631728
Average MSE, for model #4: 1.635356376757887e+16
Average RMSE: 127.88105320014716 million
```



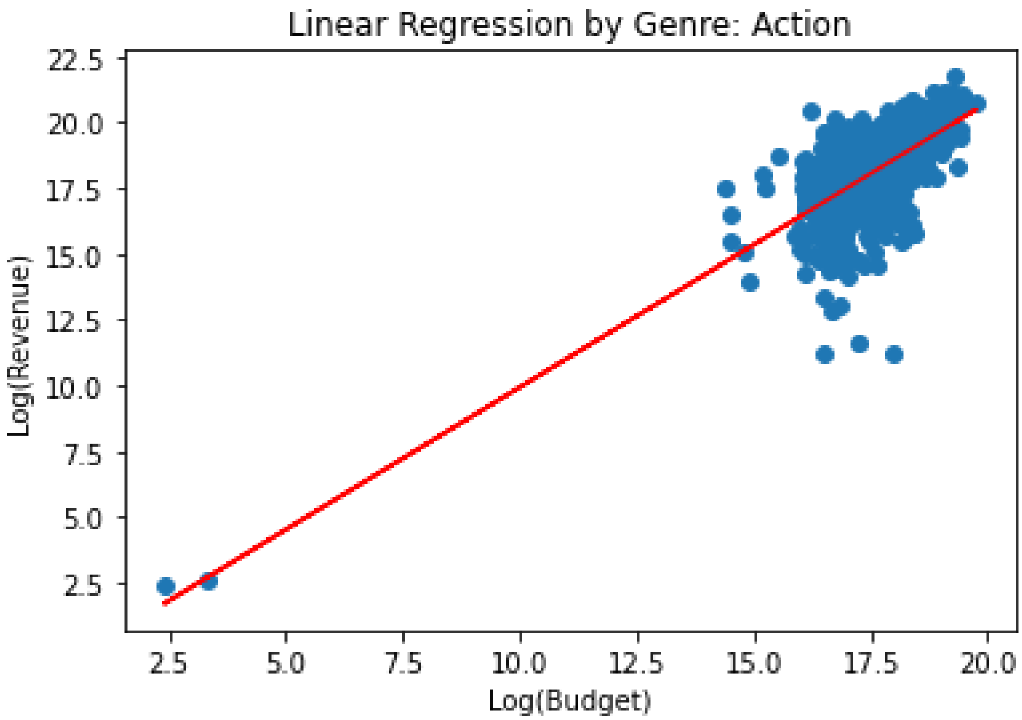

```
The Family movie linear regression model is: Revenue = 3.0168631076545847 *
Budget + -1063080.591673404
Average MSE, for model #4: 1.6790654017291932e+16
Average RMSE: 129.57875604161327 million
```



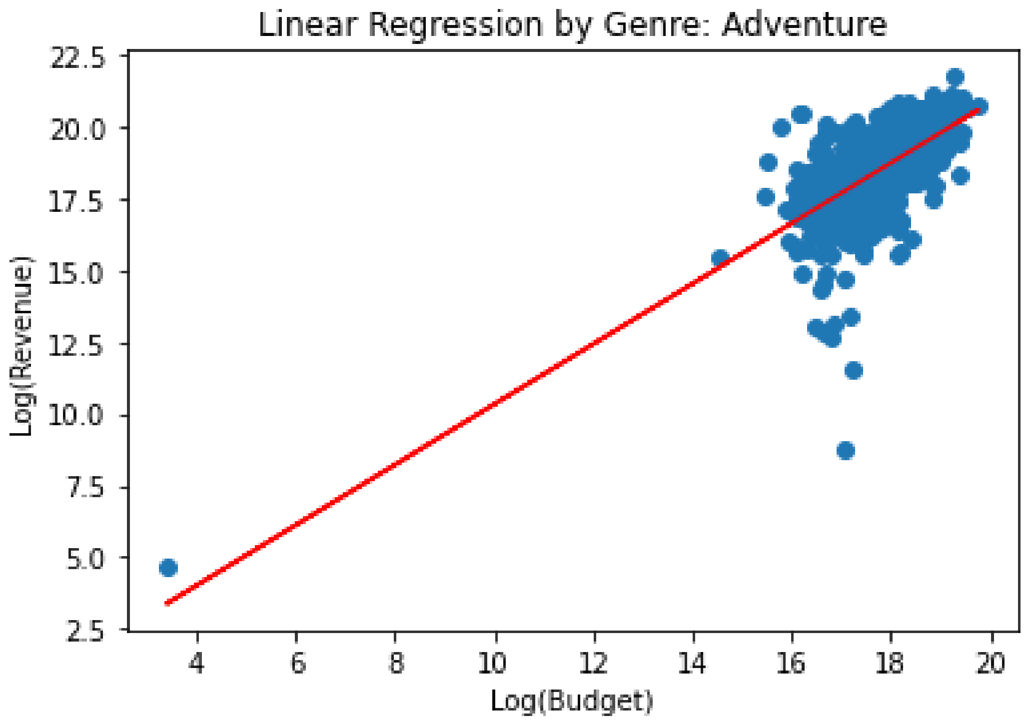

```
The Romance movie linear regression model is: Revenue = 2.968707283130696 *
Budget + 1179260.5576811433
Average MSE, for model #4: 1.6933706202477676e+16
Average RMSE: 130.129574664938 million
```




```
The Comedy movie linear regression model is: Revenue = 2.956006875647089 *
Budget + 2468763.8328278065
Average MSE, for model #4: 1.6918827347771438e+16
Average RMSE: 130.0723927194831 million
```

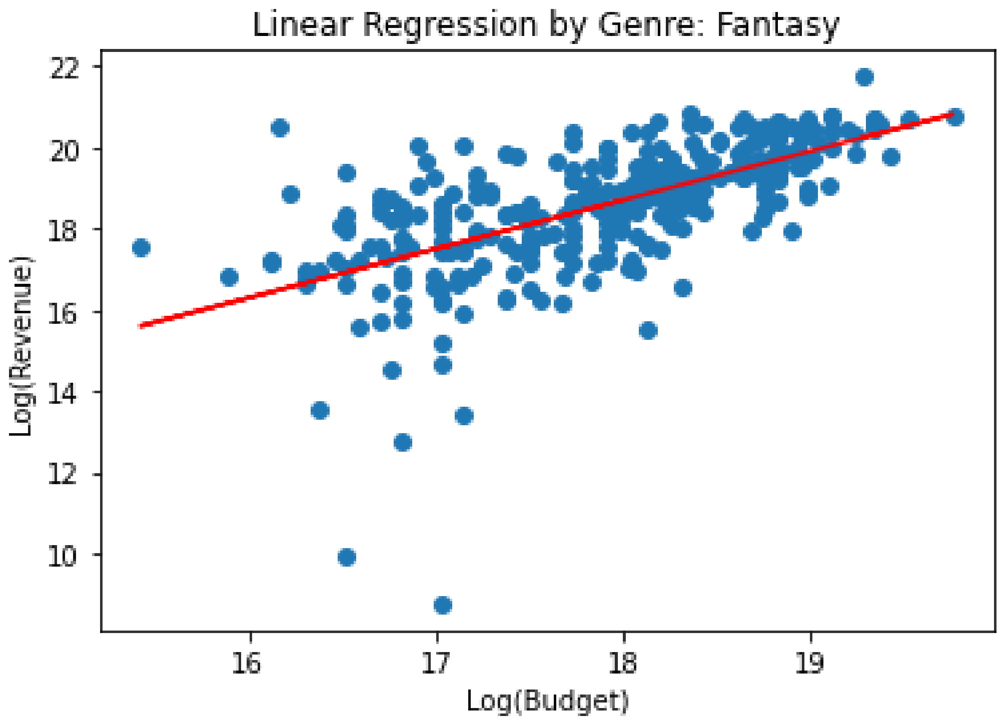



```
The Action movie linear regression model is: Revenue = 3.010564135716209 *
Budget + -166689.70156021416
Average MSE, for model #4: 1.7034759186696102e+16
Average RMSE: 130.51727543392906 million
```

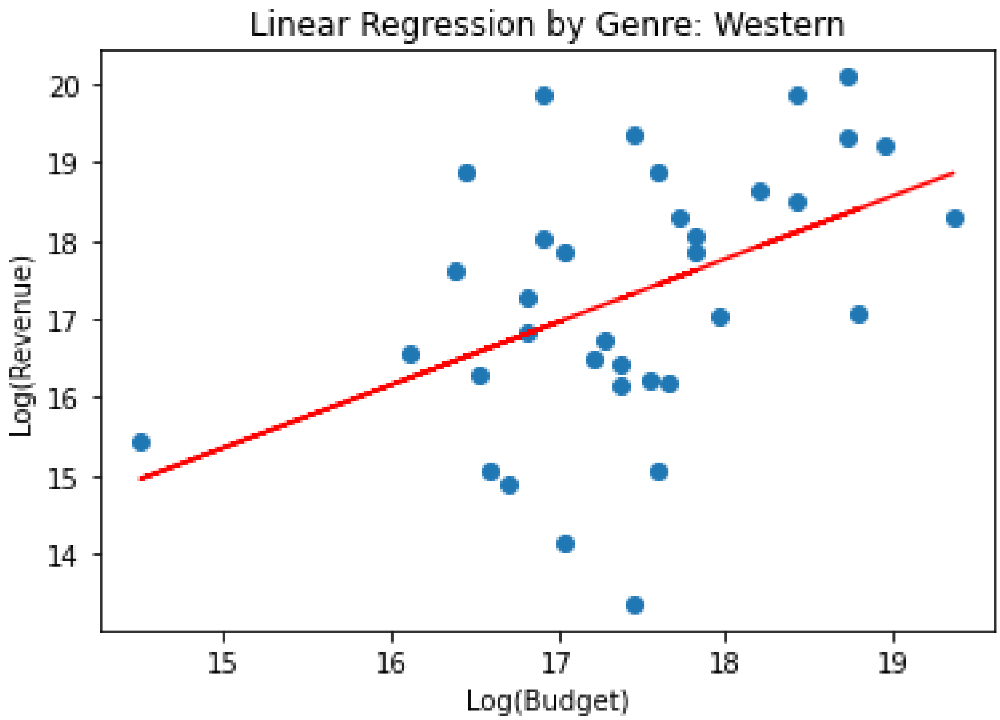


The Adventure movie linear regression model is: Revenue = 2.9120650753473956 * Budget + 1987302.8563117683
Average MSE, for model #4: 1.7323807536909798e+16
Average RMSE: 131.6199359402283 million

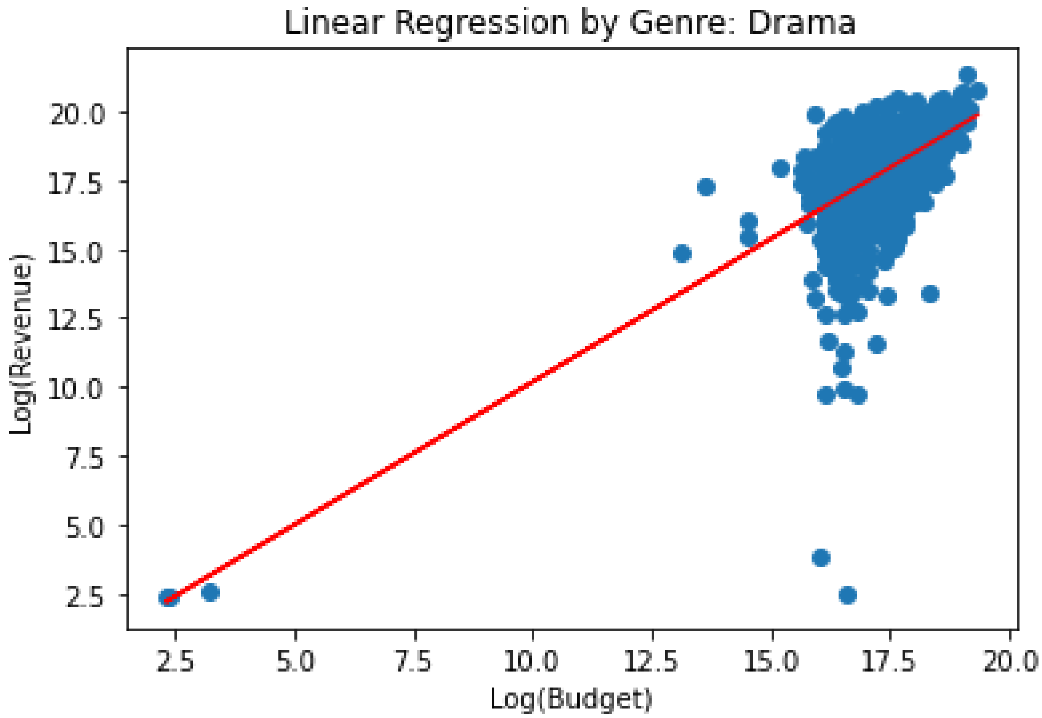
Average MSE, for model #4: 1.7323807536909798e+16
Average RMSE: 131.6199359402283 million



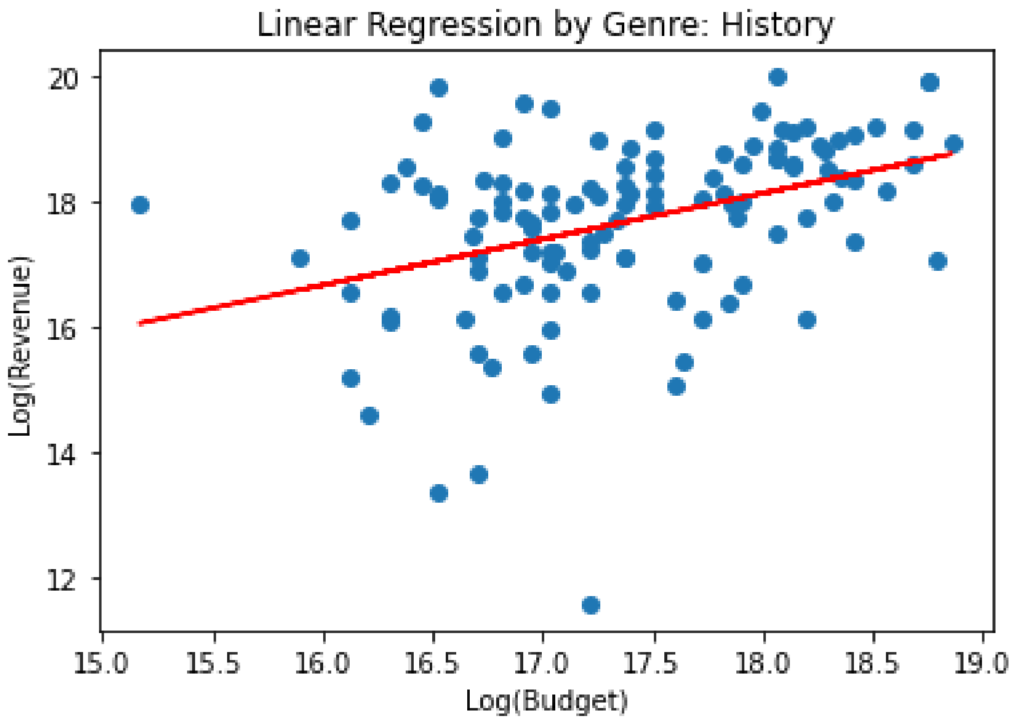
The Fantasy movie linear regression model is: Revenue = 2.9708051727239884 * Budget + 1203317.747117594
Average MSE, for model #4: 1.7350676889483732e+16
Average RMSE: 131.72196813547743 million



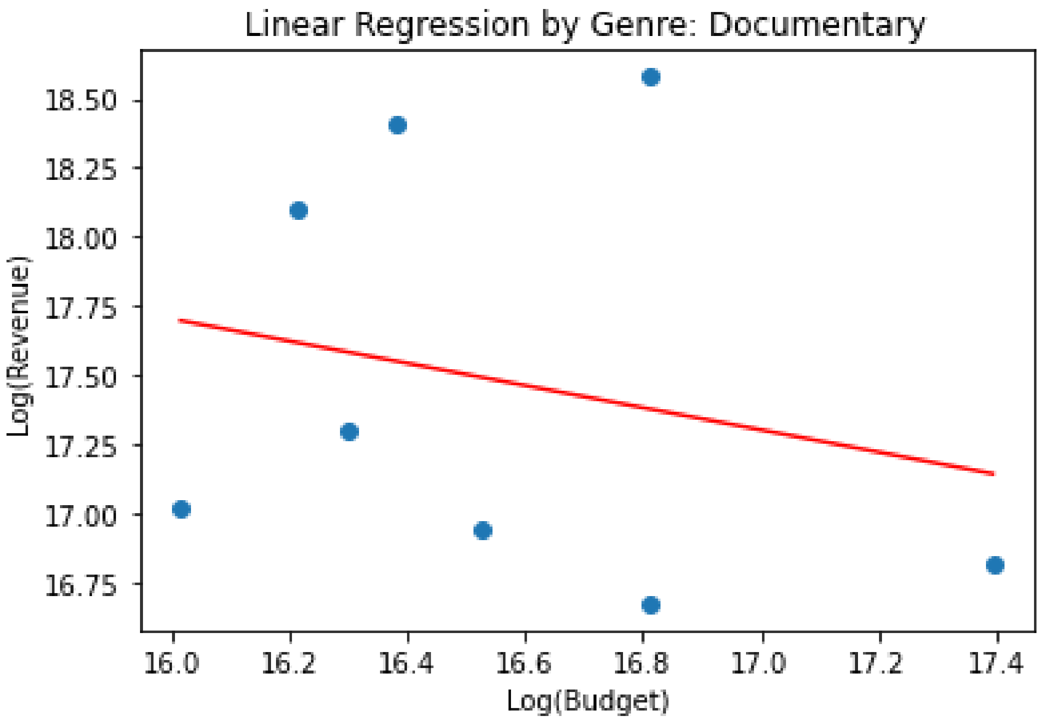

```
The Western movie linear regression model is: Revenue = 2.926058396597216 *
Budget + 1347649.497195825
Average MSE, for model #4: 1.7370247110030754e+16
Average RMSE: 131.79623329227113 million
```



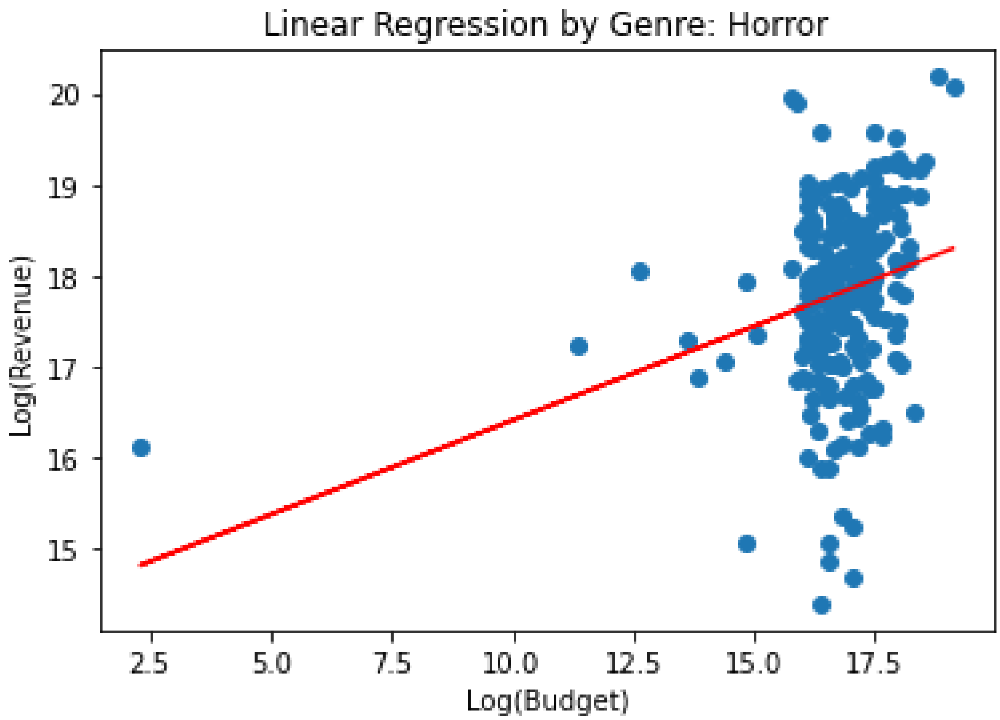

```
The Drama movie linear regression model is: Revenue = 2.8945770539482534 *
Budget + 4471614.5070246905
Average MSE, for model #4: 1.7417824419012764e+16
Average RMSE: 131.97660557467285 million
```



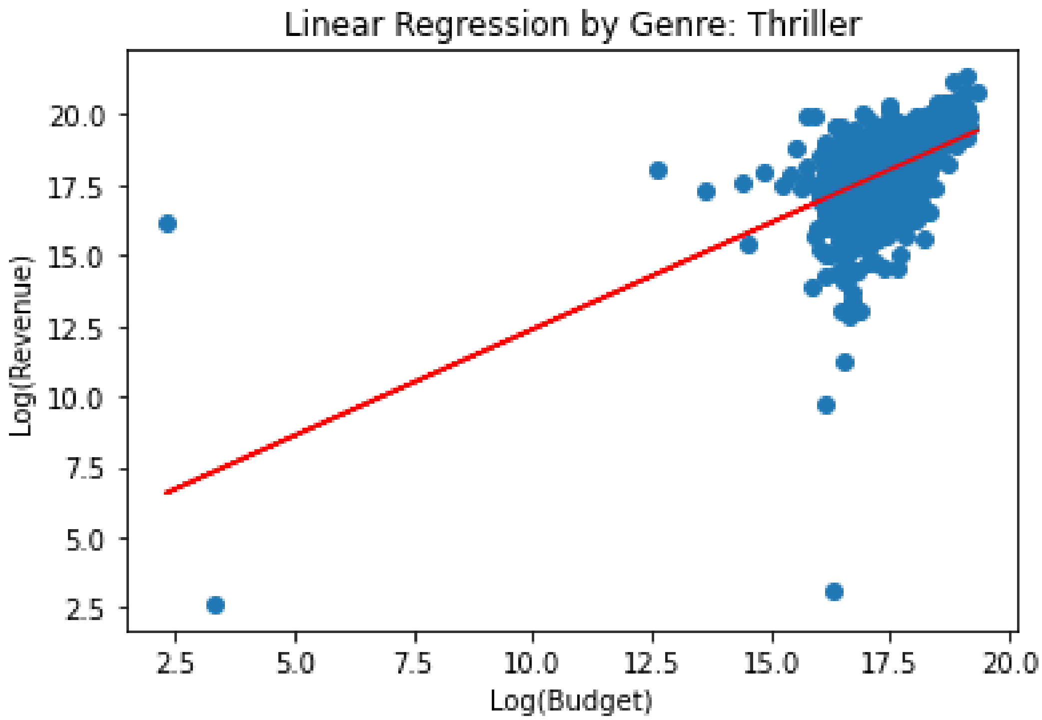
The History movie linear regression model is: Revenue = 2.85357520998209 * Budget + 3242229.65944688
Average MSE, for model #4: 1.7438709903074764e+16
Average RMSE: 132.05570757477605 million



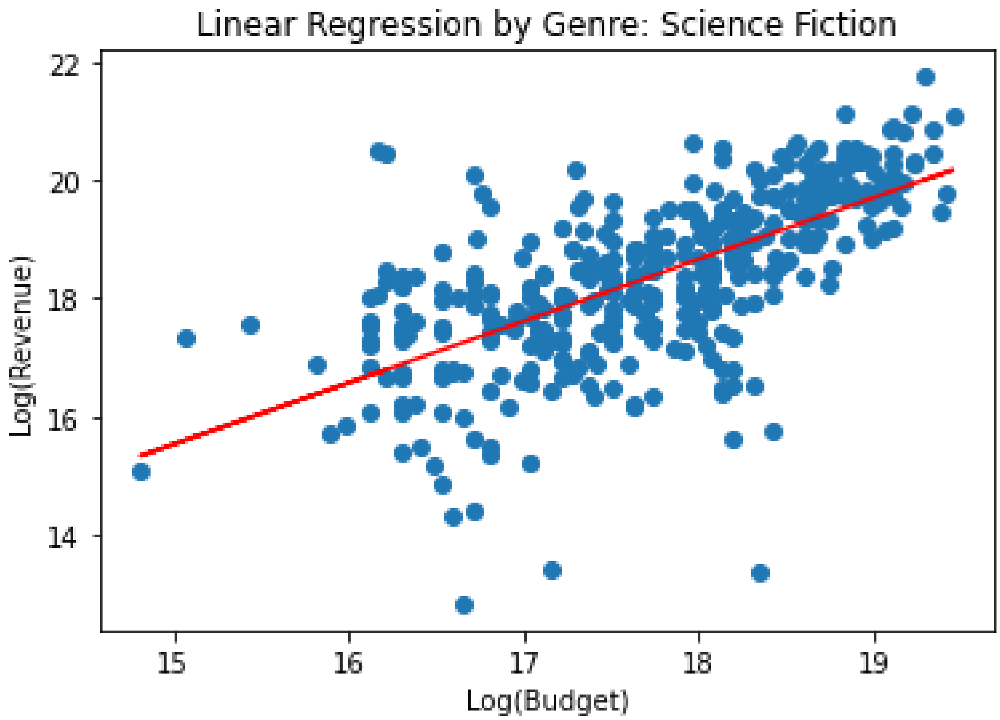

```
The Documentary movie linear regression model is: Revenue = 2.8609469964996066 *
Budget + 4541362.440034643
Average MSE, for model #4: 1.7538407446663342e+16
Average RMSE: 132.43265249425212 million
```



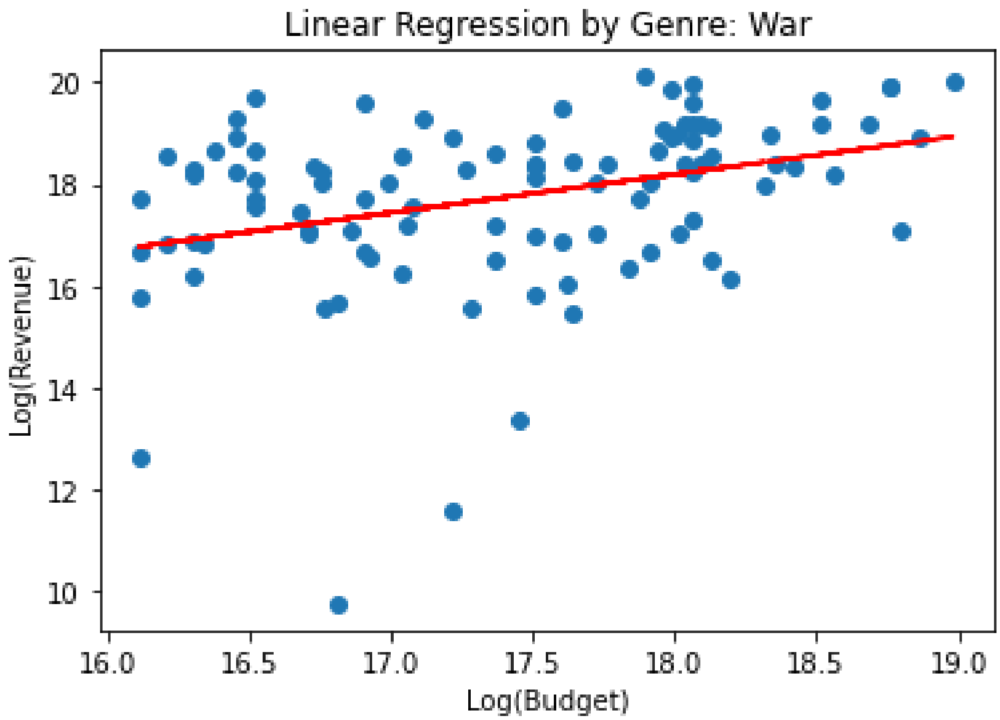
The Horror movie linear regression model is: Revenue = 2.9513514860145427 * Budget + 3702741.271984011
Average MSE, for model #4: 1.7410590618203186e+16
Average RMSE: 131.94919711086985 million



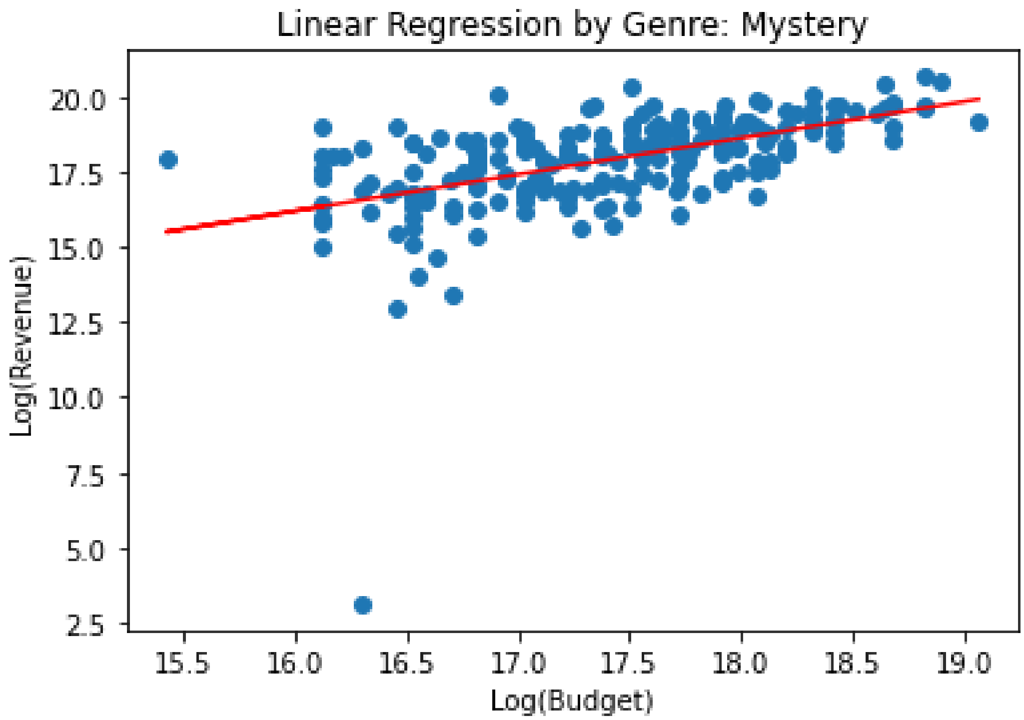

```
The Thriller movie linear regression model is: Revenue = 3.0103702779199484 *
Budget + 1409821.0873937309
Average MSE, for model #4: 1.7364205093111456e+16
Average RMSE: 131.7733094868284 million
```



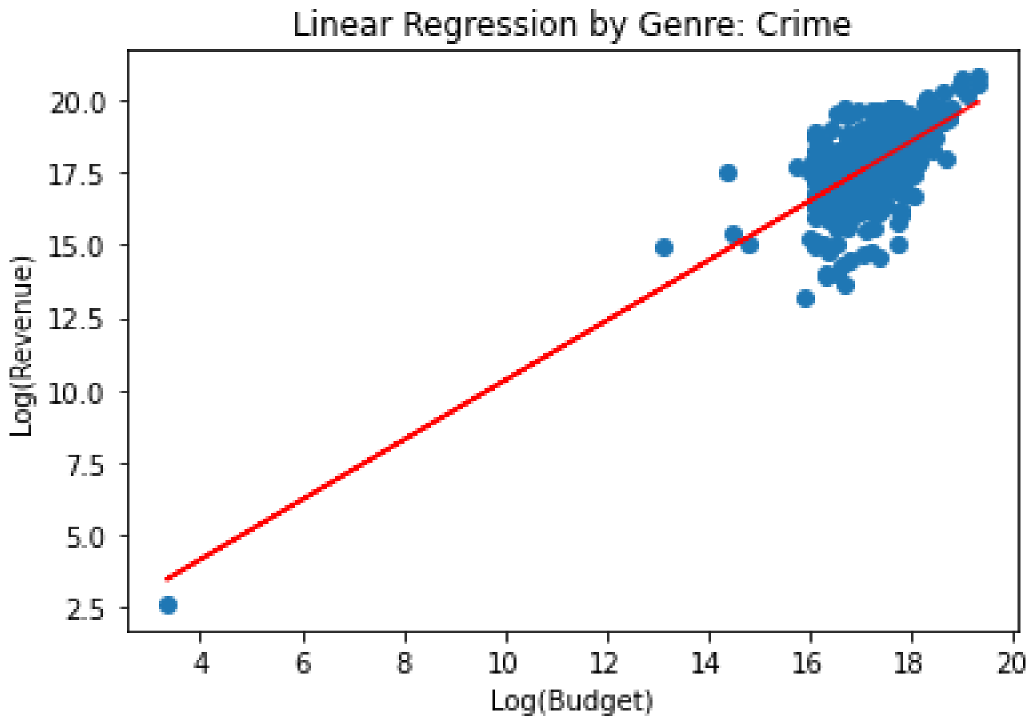

```
The Science Fiction movie linear regression model is: Revenue =
2.9470690243017135 * Budget + 1416551.0724271983
Average MSE, for model #4: 1.7381555446870426e+16
Average RMSE: 131.83912714695293 million
```




```
The War movie linear regression model is: Revenue = 3.0275604751890617 * Budget
+ -1084219.3206533939
Average MSE, for model #4: 1.7356922222817196e+16
Average RMSE: 131.74567250129013 million
```




```
The Mystery movie linear regression model is: Revenue = 3.011859642641984 *
Budget + -15542.013239726424
Average MSE, for model #4: 1.734993791424026e+16
Average RMSE: 131.71916304866298 million
```




```
The Crime movie linear regression model is: Revenue = 2.906373692474324 * Budget
+ 3234639.4060617685
Average MSE, for model #4: 1.736732000406586e+16
Average RMSE: 131.78512815968978 million

Average MSE across all genres is 1.8167708291878908e+16
Average RMSE across all genres is 138.47620884129108 million
```

Here we have cross-validation using the applications of kfolding. As a result of kfolding and cross validation, the average MSE and average RMSE has decreased slightly but not by much hence rendering this an unsuccessful altercation. The cross validation results to around 1.82e+16 and 138 million respectively.