Predictive_Model_Of_Movie_Revenue

October 30, 2018

1 Predictive Modelling of Revenues of Modern American Movies

by Stephen Gou

Oct 28, 2018

Student Number: 1000382908

1.1 Introduction

A movie's box office is the most common metric to gauge its success. A good prediction of the revenue of a movie can guide production companies for building successful movies, and inform investors to pick out the most profitable movies. This project builds a model that predicts a movie's total revenue, given certain traits and facts about the movie. Only movies produced in the United States from 1990 to 2016 are considered, because the entertainment industry and economy changes over time. Movies produced after 2016 are not considered, because have not reached their full total revenue potential. Only movies produced in the U.S are considered, because the market characteristics vary over countries and the modelling of this aspect is beyond the scope of this project. This project aims to provide effective prediction as soon as the movies are released, which means that data like opening weekend box office, IMDb rating, social media sentiments cannot be used as features in the models.

To build an effective predictive model and gain insight, the project first explores and analyzes the major factors that affect a movie's revenue. And then, a model that best suits the case will be selected and trained. Its performance will be analyzed and compared to an alternative model. Last but not least, the model's limitations and potential improvements will be discussed.

1.2 Data Collection

This project makes use of several sources to collect data for analysis and training. Various types of data are collected that includes movie's revenue, budget, meta-data, cast, crews, rankings of actors and actresses and so on. The detail of all the datasets used is listed below.

1) TMDB 5000 Movies dataset. This is the main dataset which provides budget, revenue, runtime, genre, release-date and production country data. Source: https://www.kaggle.com/tmdb/tmdb-movie-metadata

2) New York Times Review dataset. This dataset includes data like whether a movie was picked by NYT critics, and review summaries. Source: NYT API

3) TMDB 5000 Crew dataset. This dataset has detailed cast and crew information, ranging from actor to writer, for each movie. Source: https://www.kaggle.com/tmdb/tmdb-movie-metadata

4) Top Actors/Actresses Rank. This the list of a Top 1000 Actors/Actresses Ranking released by IMDb. Source: IMDb

5) Top directors Rank. This the list of a Directors Ranking released by IMDb. Source: IMDb

6) Annual CPI. This dataset lists the annual average CPI for U.S. Source: UsInflationCalculator.com

1.2.1 Cleaning

Movies produced before 1990 and after 2016 are discarded. Movies produced outside of U.S are discarded. Some movies have zero revenue in the dataset, which might be a result of missing data or unreleased movie. These movies are removed.

1.3 Feature Selection and Mapping

There are a large amount of factors that might affect a movie's revenues ranging from movies' meta-data, to unemployment rate of the release year. Features that will be analyzed and incorporated into the predictive model are selected based on availability, informativeness, unambiguity, and interpretability. According to this criteria, the following features are selected: budget, runtime, critics-pick, genres, MPAA-rating, cast, and director. The following procedures and transformations of data are done to make data representable for modelling and to increase accuracy.

1) The cast of a movie is represented by a popularity score, which is calculated by the following rule. A percentile rank score for each cast is calculated according to the actors rank dataset. Then use 1 - percentage rank as the popularity score for a cast. So 1 is the highest one can get and 0 is the lowest (0 if cast not in the ranking). Then the cast popularity for the movie is calculated as following:

$$Cast\ Popularity\ Score = \sum_{i}^{N} \gamma^{i}(1 - PercentileRank(Cast\ i))$$

   where gamma is a decay factor and N is the number of casts.

2) The director is represented by a popularity score, which is calculated by the following rule. A percentile rank score is calculated according to the directors rank dataset. Then use 1 - percentage rank as the popularity score. So 1 is the highest one can get and 0 is the lowest (0 if director not in the ranking).

3) The revenue and budget are adjusted for inflation according to the rule:

$$adjusted = \frac{CPI(2017)}{CPI(release\,year)} * unadjusted.$$

   CPI are from the Annual CPI data.

4) Genres are converted by one-hot encoding. Note that a movie can have multiple genres associated with it.

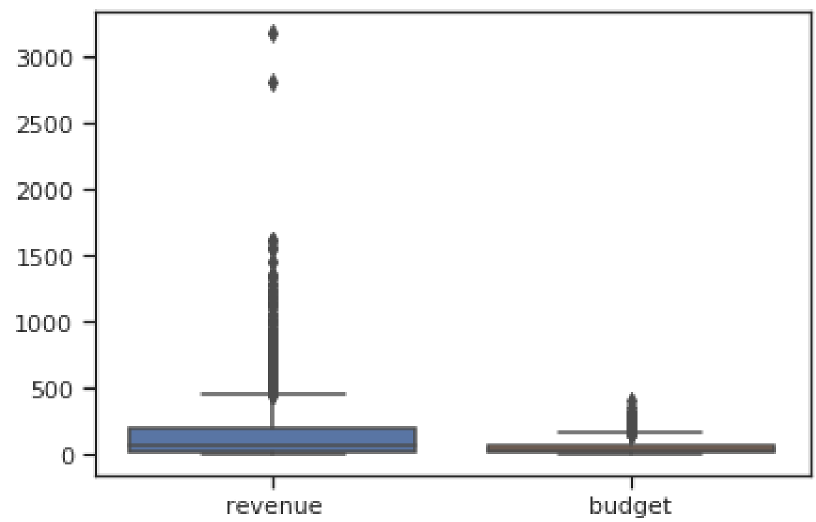
5) MPAA ratings are converted by one-hot encoding.

6) Runtime represented by a number and unchanged.

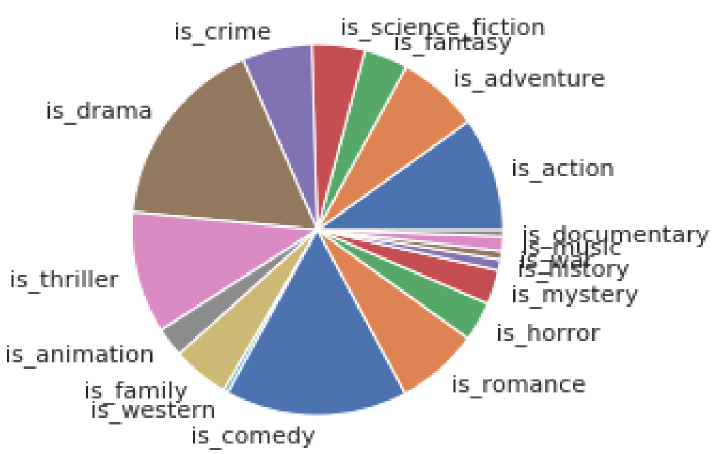
7) Critics pick is represented by 1 or 0 (1 repesents being picked)

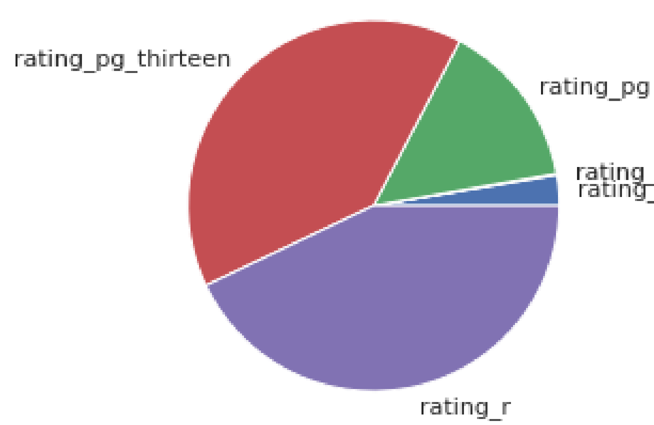
1.4 Exploratory Data Analysis

Some observations can be made from the statistics of our wrangled dataset. There are 2,033 movies in our final dataset. 17% of the movies are picked by the critics. Average runtime of a movie is 108 minutes while the lengthiest runs more than 4 hours, the shortest runs 46 mintues.



Distributions of Data Revenues and budgets of movies are concentrated in low values, \$ 78m and \$ 37m respectively. There are large number of outliers in both cases. However, revenue has very long-tail towards higher values and outliers with more extreme values.





There are quite diverse and evenly distributed number of genres in the data. And majority of movies are at least PG-13.

Correlations Between Features A heatmap of correlation between features is plotted to spot features that have strong relationships with each other, so that redundant features can be discarded to reduce multicollinearity.

Genres and mpaa-rating tend to have strong correlations. From the plot, it's clear that movies that have "family" as a genre is also very likely to have "animation" as a genre as well. Family and animation movies also usually have PG or G rating.

The quality of the cast appear to be uncorrelated with most of the genres of movies except for horror, where quality of cast drops significantly.

Intuitively, the runtime of a movie has correlations with its genres, which is confirmed by the heatmap. The runtime also correlates with budget and quality of director and cast.

Another interesting observation is that New York Time's critics' picks appear to be uncorrelated with most of the features of a movie, meaning that they are not favoring a particular subsets of movies over the others. Action and thriller movies are marginally less likely to be picked, but that could just be a result of noise.

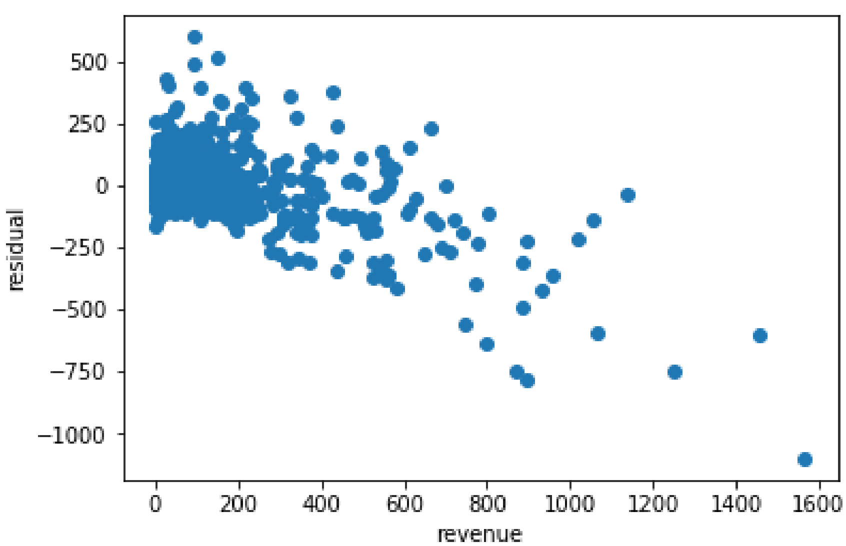
From these observations, runtime and mpaa-rating of a movie could be potentially discarded, because they usually depend on other features of the movie.

1.5 Analysis and Modelling

Since the goal is to predict revenue, a continuous value over a wide range, regression models are considered. More specifically, OLS regression, Ridge/Lasso regression, Regression Tree, Random Forest regression and Multilayer Perceptrons are the candidate models.

Initially, the models' performance are evaluated based on the R-Squared statistic and the residual plot. An OLS regression model that simply includes all the features without adding higher order terms and interactions is fitted and its result is used as a baseline. Model is trained on training set, which is 70% of the dataset.

It obtained a **R-Squared score of 0.492** on the test set and the residual plot as below:



The other models obtained similar results when fitted with only linear terms. Firstly, the R-Squared statistic is not very informative, given that revenue is an unbounded number and there are large number of outliers in the dataset. Secondly, the residual plot displays a clear linear relationship between residual and the revenues of the movies, meaning that there is a significant pattern of revenues of movies that it is not uncovered yet. However, there is inherently large

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                revenue   R-squared:                       0.465
Model:                            OLS   Adj. R-squared:                  0.463
Method:                 Least Squares   F-statistic:                     246.3
Date:                Tue, 30 Oct 2018   Prob (F-statistic):          1.53e-189
Time:                        15:35:38   Log-Likelihood:                -9395.3
No. Observations:                1424   AIC:                         1.880e+04
Df Residuals:                    1418   BIC:                         1.883e+04
Df Model:                           5
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept       -103.6427     29.681     -3.492      0.000    -161.866     -45.419
runtime            0.8579      0.296      2.899      0.004       0.277       1.438
budget             2.9154      0.097     30.023      0.000       2.725       3.106
critics_pick      32.4342     14.170      2.289      0.022       4.638      60.230
cast_score         3.7086      5.703      0.650      0.516      -7.479      14.896
director_score    31.1282     24.247      1.284      0.199     -16.436      78.692
==============================================================================
Omnibus:                     1225.181   Durbin-Watson:                   2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            67372.728
Skew:                           3.679   Prob(JB):                         0.00
Kurtosis:                      35.884   Cond. No.                         835.
==============================================================================
```

uncertainly of movie's revenue and given limited information there are about the movies. It's necessary to find a more effective metric to evaluate the models.

The alternative metric is defined as the percentage of predictions that are within 20% error from the true value. It will be referred to as "accuracy". The baseline is **34.8%**, which is obtained by simply use the mean revenue for every prediction.

1.5.1 OLS Linear Regression

Initially, a brute force model that includes a large number of interactions between between features, and certain second order terms (102 total terms in regression formula) is fitted. It obtained accuracy of **49.6%**.

The following steps are taken to improve the performance, interpretability and reduce overfitting.

- Like suggested in EDA, mpaa-rating is discarded because it depends on other features.
- Genres are discarded as well. OLS regression shows an extremely large condition number (> 10ˆ 10) with genres included, meaning there are strong multi-collinearity. In addition, it caused certain terms to have extremely large weights, even when L1/L2 reguarizers are added. Lastly, in introduced too many potential interactions between each other and other features like directors, actors and budget.
- Naturally removing outliers from dataset was considered, but removing them did not improve any model's performances. Therefore, outliers are kept.

The resulting formula for regression is

$$revenue = Intercept + w_0 * runtime + w_1 * budget + w_2 * criticspick + w_4 * castscore + w_5 * directorscore$$

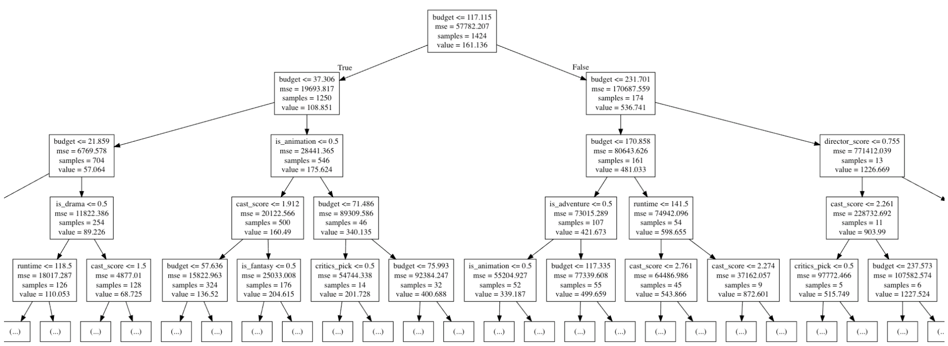
It achieved an accuracy of **55.8%**. A 6% increase comparing to the brute force model. Runtime, budget, and critics pick all have coefficients with lower than 5% p-value, and the coefficents have large values. All else equal, a movie makes \$ 32m more than it's picked by NYT critics. For every million dollar spent on budget, there are \$ 2.9m revenue in return. Actors score and director score have coefficients with large P-value, meaning that it cannot be concluded that a great cast or director will surely drive up revenue.

1.5.2 Non-Linear Regression Models

Given the large number of potential interactions and non-linear relationship between certain features and revenue, it is extremely hard manually select features. Thus, non-linear models like regression tree and multilayer perceptron (neutral networks with only fully connected hidden layers) are considered. For these models, all available features are included in the input.

1) A MLP with two hidden layers of size 5 and 3 with ReLU activations is trained for 2,000 iterations achieved accuracy scores ranging from **47.0%** to **54.0%**

2) A regression tree with unlimited depth is constructed and achieved accuracy score of **60.7%** The top few layers of the tree is shown as below:



Budget seems to provide the most information gain, as itappear most frequently in top layers. More interestingly, certain genres, seem to have significant effects. For example, in lower budget movies (budget < \$ 117m) animation movies average \$ 340m revenue while the others only average \$ 160m. For high budget movies (more than\$ 230), directors that rank in top 25% average \$ 3000m revenue while others only \$ 903m.

1.6 Results

This project has constructed a dataset of American movies produced between 1990 and 2016 that include features: budget, runtime, genres, mpaa_rating, cast_score, and director_score. The best linear model achieved **55.8%** accuracy (with 20% error allowed) and the best non-linear model, which is a regression tree achieved **60.7%**. As a reference, predicting the mean everytime will score **34.8%**.

The linear model shows that runtime, budget and critics pick have significant contributions to the gross revenue of a movie. And the decision tree shows that budget provides the most information gain. In addition, the tree shows that genres' effects and director's effect depend on other features, such as budget.

1.7 Future Work and Improvement

Find better feature data Since the residual shows that there is a significant linear pattern for the revenues missed, more relevant data and data that have more explaining power might be explored and incorporated into the models. For example, the number of views of movie's trailers before release, social media influence of casts, writer of the movie, production company and so on. Another possibility is to loosen the assumption so that more post-release data can be incoporated like opening weekend box office, IMDb rating, hashtag counts and so on.

Improve existing features There are room for improvements of the features currently used in the models. For example, how the cast and director score is calculated could be improved. Instead of rank based, maybe include more revenue related traits for example, actors' social media following, revenues of past 3 movies, and so on.

Improve modelling Since the models are systemically predicting overestimated revenue for low revenue movies and underestimated revenue for high revenue movies, there might be opportunity to take advantage of this observation. For example, use locally weighted regression, kNN or an ensemble of models so that movies in differernt levels can be modelled separately.

1.8 Python Code

```
In [1]: import json
        import requests
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns;
        from pandas.io.json import json_normalize
        import statsmodels.api as sm
        import statsmodels.formula.api as smf
        from sklearn.model_selection import train_test_split
        from sklearn import linear_model
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neural_network import MLPClassifier
        from sklearn.neural_network import MLPRegressor
        from sklearn.ensemble import AdaBoostRegressor
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn import tree
```

1.8.1 Cleaning and Feature Mapping

```
In [2]: github_raw_root = 'https://raw.githubusercontent.com/gouzhen1/Moives-Data-Analysis/mas

        #NY Reviews Dataset
        ny_df = pd.read_csv(github_raw_root + 'NY_movie_reviews.csv')
        ny_df.rename(columns={'display_title':'title'},inplace=True)
        ny_df = ny_df[['title','mpaa_rating','critics_pick']]
```

```
#Wrangle actors and director
#TMDB Credits Dataset (for cast and director)
tmdb_credits_df = pd.read_csv(github_raw_root + 'tmdb_5000_credits.csv')
actors_rank = pd.read_csv(github_raw_root + 'Top_actors_rank.csv')['Name'].tolist()
directors_rank = pd.read_csv(github_raw_root +'All_time_director_rank.csv')['Name'].to
total_actors = len(actors_rank)
total_directors = len(directors_rank)

def transform_cast(df):
    cast_json = df['cast']
    parsed_cast = json.loads(cast_json)
    score = 0.
    count = 0
    for cast in parsed_cast:
        actor = cast['name']
        if actor in actors_rank:
            #discounted for later casts
            score += (0.9 ** count) * (1. - (actors_rank.index(actor)/total_actors))
        count += 1
    return score
tmdb_credits_df['cast_score'] = tmdb_credits_df.apply(transform_cast, axis = 1)

def transform_crew(df):
    crew_json = df['crew']
    parsed_crew = json.loads(crew_json)
    score = 0.
    for crew in parsed_crew:
        if crew['department'] == 'Directing' and crew['job'] == 'Director':
            director = crew['name']
            if director in directors_rank:
                score += (1. - (directors_rank.index(director)/total_directors))
            break
    return score

tmdb_credits_df['director_score'] = tmdb_credits_df.apply(transform_crew, axis = 1)
tmdb_credits_df = tmdb_credits_df[['title','cast_score','director_score']]

#TMDB Main Dataset
main_df = pd.read_csv(github_raw_root + 'tmdb_5000_movies.csv')
main_df['release_date'] = pd.to_datetime(main_df['release_date'])
main_df.drop(main_df[main_df['release_date'].dt.year < 1990].index, inplace=True)
main_df.drop(main_df[main_df['release_date'].dt.year > 2016].index, inplace=True)
main_df = main_df[main_df['revenue'] > 0]
main_df = main_df.merge(ny_df,how='left')

#process and filter countries
def process_country(df):
    country_json = df['production_countries']
```

```
    parsed_country = json.loads(country_json)
    if len(parsed_country) > 0:
        return parsed_country[0]['name']
    else:
        return None
main_df['production_countries'] = main_df.apply(process_country, axis = 1)
main_df = main_df[main_df['production_countries'] =='United States of America']
main_df.drop(columns='production_countries',inplace=True)

#wrangle genre
genre_dict = {}
def transform_genre(df):
    genre_json = df['genres']
    parsed_genre = json.loads(genre_json)
    result = []
    for genre in parsed_genre:
        genre_name = genre['name'].replace(' ','_')
        result.append(genre_name)
        if genre_name not in genre_dict:
            genre_dict[genre_name] = 1
        else:
            genre_dict[genre_name] += 1

    return result
main_df['genres'] = main_df.apply(transform_genre, axis = 1)
#drop very low rare genres
del genre_dict['Foreign']
for genre in genre_dict:
    main_df['is_' + genre] = main_df['genres'].transform(lambda x: int(genre in x))
main_df.drop(columns=['genres'],inplace=True)

#map mpaa rating
rating_df = pd.get_dummies(main_df['mpaa_rating'],prefix='rating')
main_df = main_df.merge(rating_df,left_index=True,right_index=True)
main_df.drop(columns=['mpaa_rating','rating_Not Rated'],inplace=True) #drop one catego

#adjust revenue and budget for inflation
cpi_df = pd.read_csv(github_raw_root + 'Annual_CPI.csv')
cpi_df = cpi_df.set_index('DATE')
cpi_dict = cpi_df.to_dict()['CPIAUCSL']
def get_cpi_adjusted_revenue(df):
    year = df['release_date'].year
    revenue = df['revenue']
    return cpi_dict['2017-01-01']/cpi_dict['{}-01-01'.format(year)] * revenue

def get_cpi_adjusted_budget(df):
    year = df['release_date'].year
    budeget = df['budget']
```

```
        return cpi_dict['2017-01-01']/cpi_dict['{}-01-01'.format(year)] * budeget

    main_df['revenue'] = main_df.apply(get_cpi_adjusted_revenue,axis=1)
    main_df['budget'] = main_df.apply(get_cpi_adjusted_budget,axis=1)
    main_df['revenue'] = main_df['revenue'] * 0.000001
    main_df['budget'] = main_df['budget'] * 0.000001

    def cat_revenue(df):
        rev = df['revenue']
        c = min(int(rev/250. * 10.),10)
        return c

    #main_df['revenue'] = main_df.apply(cat_revenue,axis=1)

    main_df = main_df.drop(columns = ['release_date','original_language','popularity','home
```

In [3]:

```
print('wrangled dataset: ' + str(main_df.shape))
main_df = main_df.merge(tmdb_credits_df,how='left')
main_df.columns = map(str.lower, main_df.columns)
main_df.rename(columns={'rating_pg-13':'rating_pg_thirteen','rating_nc-17':'rating_nc_s
main_df['critics_pick'].fillna(0,inplace=True)
main_df.to_csv('wrangled_dataset.csv')
main_df.rename(columns={'rating_not rated':'rating_not_rated'},inplace=True)
main_df.head()
```

```
wrangled dataset: (2033, 28)
```

Out[3]:

```
       budget      revenue  runtime                                    title  \
0  270.771526  3185.238655    162.0                                   Avatar
1  354.684562  1136.172881    169.0  Pirates of the Caribbean: At World's End
2  266.936095  1158.437625    165.0                    The Dark Knight Rises
3  277.613539   303.387927    132.0                              John Carter
4  305.028724  1053.261376    139.0                             Spider-Man 3

   critics_pick  is_action  is_adventure  is_fantasy  is_science_fiction  \
0           1.0          1             1           1                   1
1           0.0          1             1           1                   0
2           1.0          1             0           0                   0
3           0.0          1             1           0                   1
4           0.0          1             1           1                   0

   is_crime     ...       is_war  is_music  is_documentary  rating_g  \
0         0     ...            0         0               0         0
1         0     ...            0         0               0         0
2         1     ...            0         0               0         0
3         0     ...            0         0               0         0
4         0     ...            0         0               0         0
```

```
   rating_nc_seventeen  rating_pg  rating_pg_thirteen  rating_r  cast_score  \
0                    0          0                   1         0    1.412578
1                    0          0                   1         0    2.403865
2                    0          0                   1         0    4.555502
3                    0          0                   1         0    1.787542
4                    0          0                   1         0    2.124545

   director_score
0        0.836364
1        0.400000
2        0.709091
3        0.000000
4        0.490909

[5 rows x 30 columns]
```

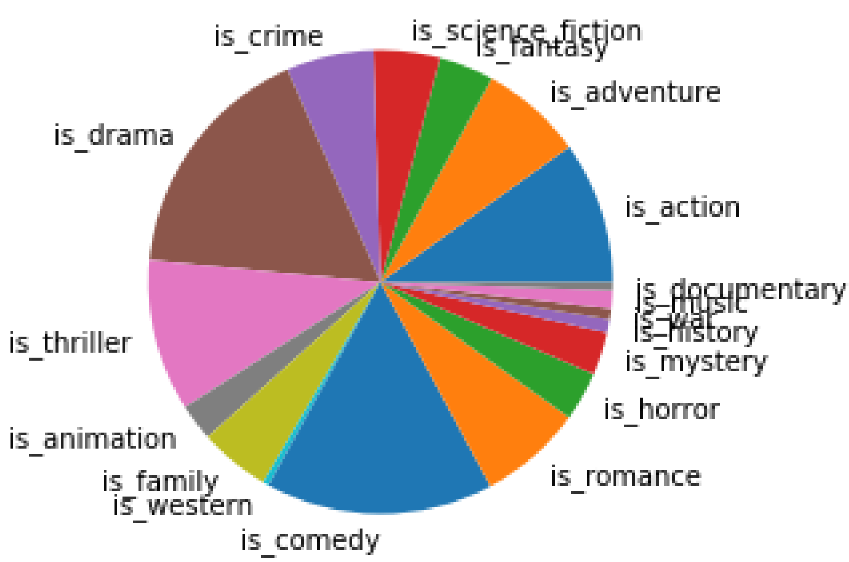
1.9 EDA

In [4]:
```
main_df.describe()
```

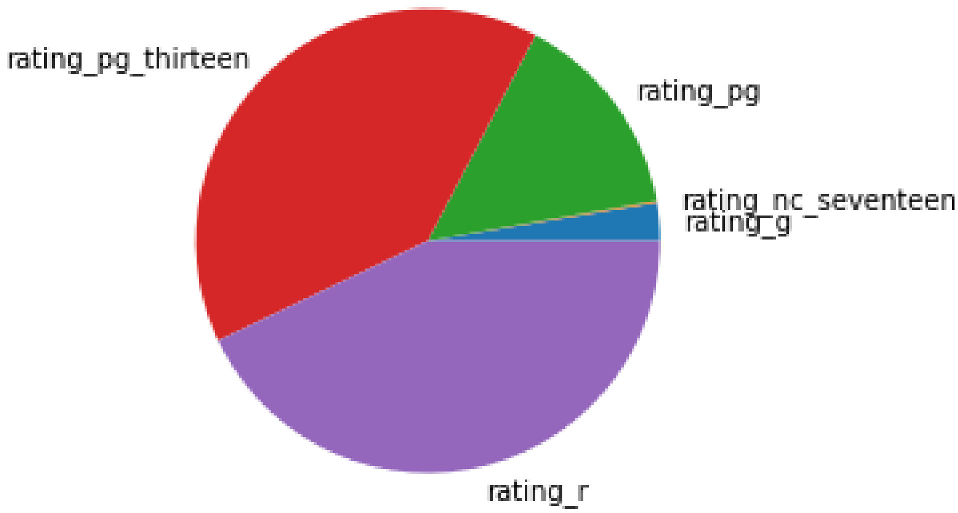
Out[4]:
```
            budget      revenue      runtime  critics_pick    is_action  \
count  2035.000000  2035.000000  2035.000000   2035.000000  2035.000000
mean     54.405731   162.734096   108.094840      0.138084     0.258968
std      53.526064   238.730589    18.560045      0.345072     0.438176
min       0.000000     0.000015    46.000000      0.000000     0.000000
25%      16.446526    25.164840    95.000000      0.000000     0.000000
50%      37.632211    78.458092   105.000000      0.000000     0.000000
75%      77.706677   198.331272   118.000000      0.000000     1.000000
max     414.154688  3185.238655   254.000000      1.000000     1.000000

       is_adventure   is_fantasy  is_science_fiction     is_crime  \
count   2035.000000  2035.000000         2035.000000  2035.000000
mean       0.186241     0.099754            0.120885     0.158722
std        0.389397     0.299746            0.326073     0.365507
min        0.000000     0.000000            0.000000     0.000000
25%        0.000000     0.000000            0.000000     0.000000
50%        0.000000     0.000000            0.000000     0.000000
75%        0.000000     0.000000            0.000000     0.000000
max        1.000000     1.000000            1.000000     1.000000

          is_drama     ...           is_war     is_music  is_documentary  \
count  2035.000000     ...      2035.000000  2035.000000     2035.000000
mean      0.441769     ...         0.019165     0.032924        0.014742
std       0.496720     ...         0.137137     0.178481        0.120548
min       0.000000     ...         0.000000     0.000000        0.000000
25%       0.000000     ...         0.000000     0.000000        0.000000
50%       0.000000     ...         0.000000     0.000000        0.000000
```

```
75%       1.000000      ...          0.000000     0.000000         0.000000
max       1.000000      ...          1.000000     1.000000         1.000000

            rating_g  rating_nc_seventeen    rating_pg  rating_pg_thirteen  \
count    2035.000000          2035.000000  2035.000000         2035.000000
mean        0.019656             0.000983     0.108600            0.296806
std         0.138849             0.031342     0.311213            0.456963
min         0.000000             0.000000     0.000000            0.000000
25%         0.000000             0.000000     0.000000            0.000000
50%         0.000000             0.000000     0.000000            0.000000
75%         0.000000             0.000000     0.000000            1.000000
max         1.000000             1.000000     1.000000            1.000000

            rating_r   cast_score  director_score
count    2035.000000  2035.000000     2035.000000
mean        0.320393     1.320750        0.072986
std         0.466742     0.963622        0.209531
min         0.000000     0.000000        0.000000
25%         0.000000     0.536000        0.000000
50%         0.000000     1.214380        0.000000
75%         1.000000     1.969573        0.000000
max         1.000000     4.644810        1.000000

[8 rows x 29 columns]
```

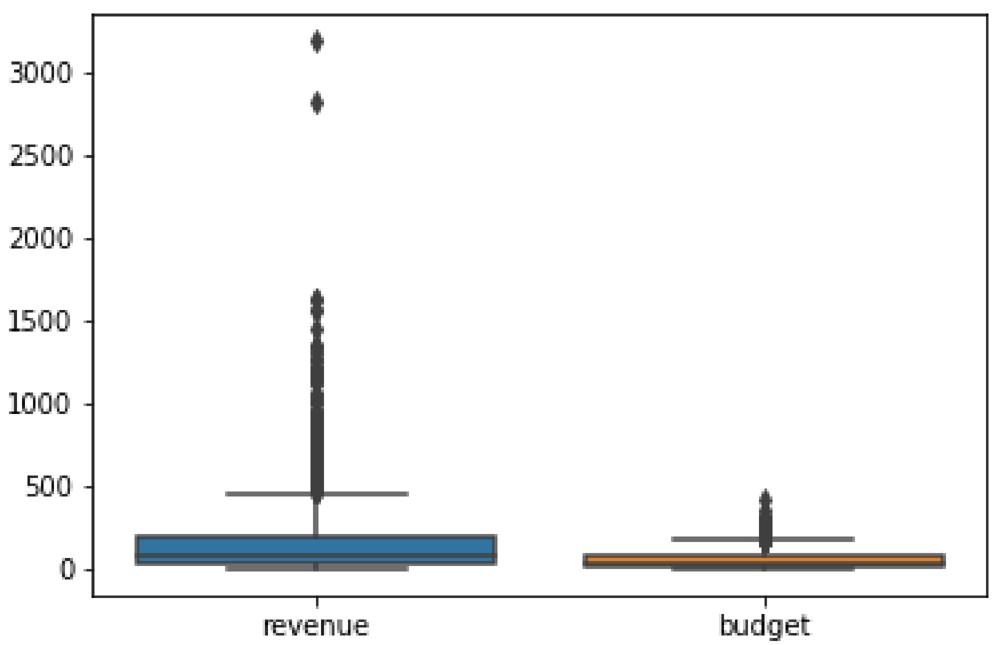
In [5]:
```
sum_df = main_df.apply(np.sum,axis = 0)
#genres pie chart
plt.pie(sum_df[5:-7],labels = sum_df.index[5:-7])
plt.show()
```



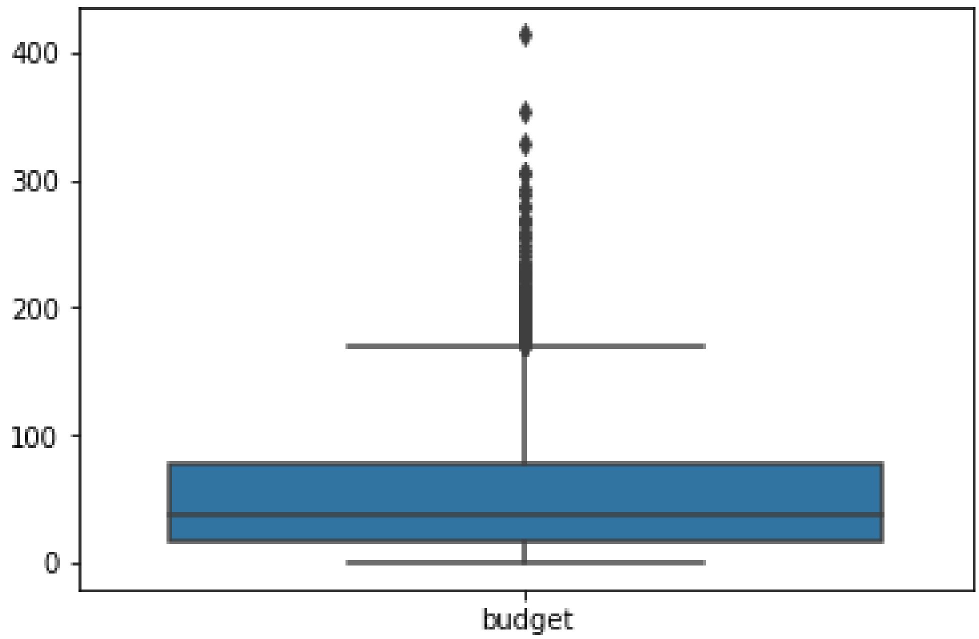
In [6]:
```
#mpaa ratings pie chart
plt.pie(sum_df[-7:-2],labels = sum_df.index[-7:-2])
plt.show()
```



In [7]:
```
sns.boxplot(data = main_df[['revenue','budget']])
plt.show()
```



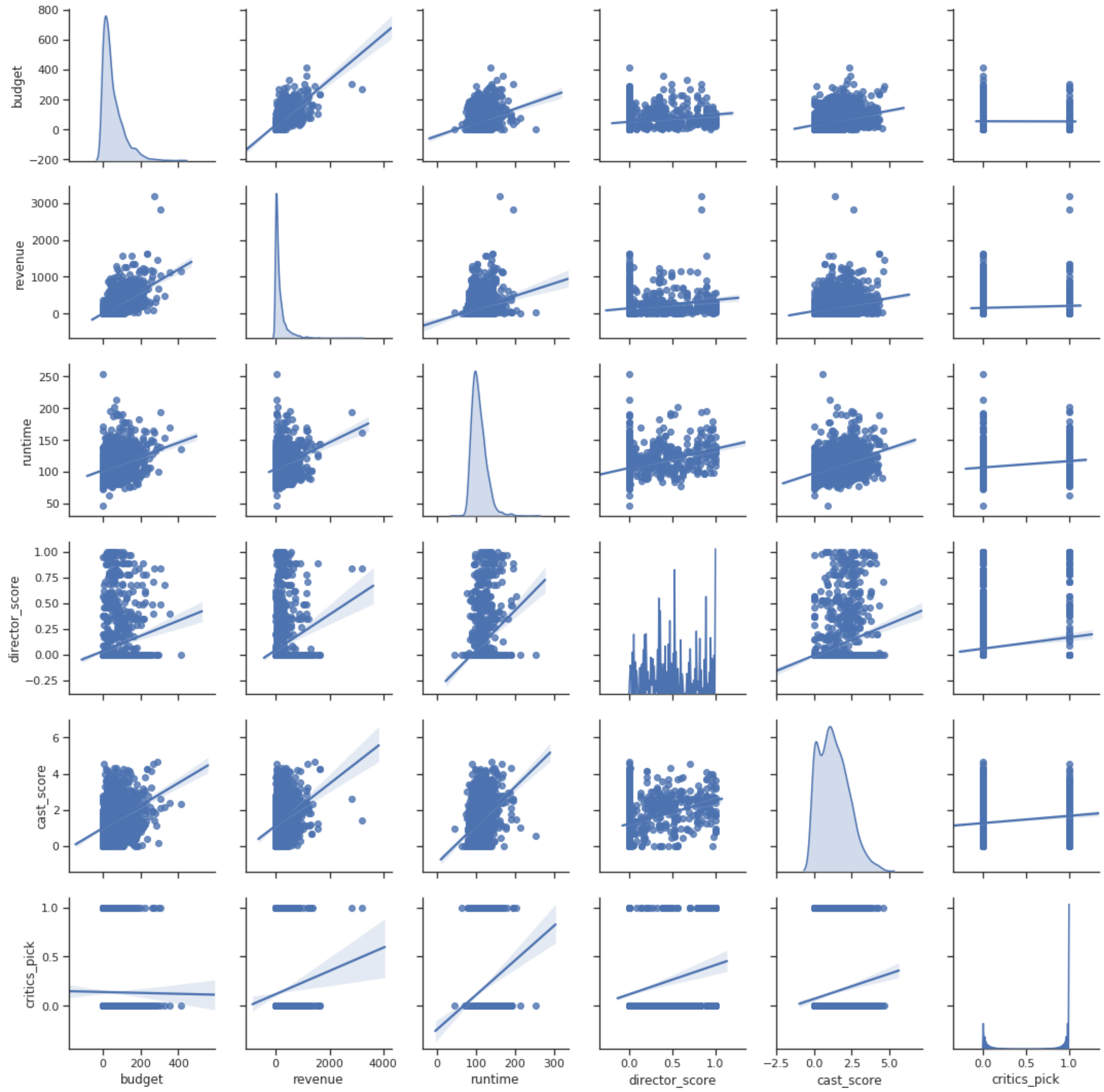
In [8]:
```
sns.boxplot(data = main_df[['budget']])
plt.show()
```



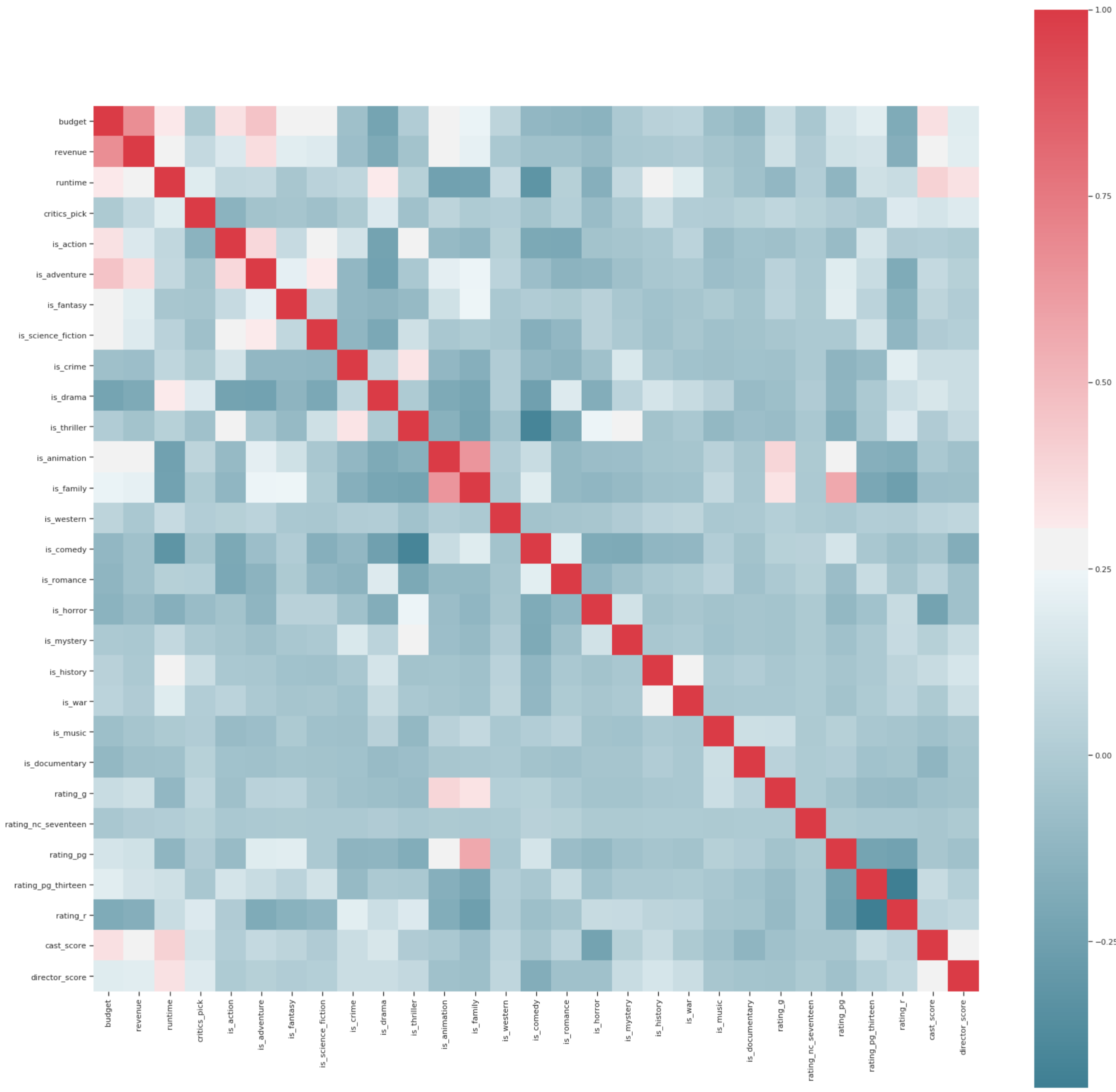
In [9]:
```
sns.set(style="ticks", color_codes=True)
#plt.scatter(main_df['budget'],main_df['revenue'])
sns.pairplot(main_df[['budget','revenue','runtime','director_score','cast_score','crit
```

```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tu
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1c267044e0>
```




```
In [10]: corr = main_df.corr()
         plt.figure(figsize = (28,28))
         sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1c27eda6d8>
```



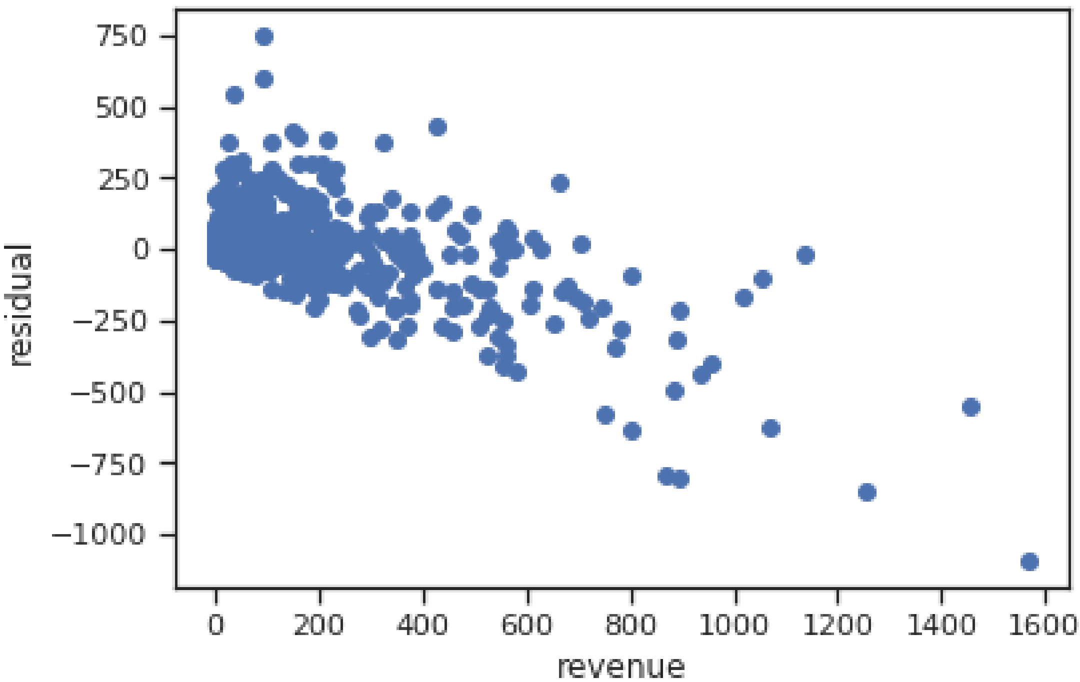
1.10 Modelling

In [11]:
```
x_df = main_df.drop(columns=['revenue','title'])
x_df.drop(columns=['rating_g' ,'rating_nc_seventeen', 'rating_pg', 'rating_pg_thirteen
y_df = main_df['revenue']
x_train,x_test,y_train,y_test = train_test_split(x_df,y_df,test_size = 0.3,random_sta
x_train.head()
ols_train = x_train.copy()
ols_train['revenue'] = y_train
limit = 0.2 #error allowed

#bench mark against just predicting mean
error_rate = (y_df.mean() - y_df)/y_df
print('total in test set: ' +str(len(y_df)))
```

```python
valid_predictions = np.sum(error_rate < limit)
print('percent with less than {} error: '.format(limit) + str(valid_predictions))
print('accuracy: ' + str(valid_predictions/len(y_df)))

def evaluate_ols_results(results):
    print(results.summary())
    predictions = results.predict(x_test)
    plt.scatter(y_test, predictions - y_test)

    error_rate = (predictions - y_test)/y_test
    correct_predictions = np.sum(error_rate < limit)
    print(correct_predictions, correct_predictions/len(y_test))
    plt.xlabel('revenue')
    plt.ylabel('residual')
    plt.show()

formula = '''revenue ~ runtime + budget + critics_pick + cast_score + director_score'

results = smf.ols(formula, data=ols_train).fit()
evaluate_ols_results(results)
```

```
total in test set: 2035
percent with less than 0.2 error: 708
accuracy: 0.34791154791154794
                            OLS Regression Results
==============================================================================
Dep. Variable:                revenue   R-squared:                       0.456
Model:                            OLS   Adj. R-squared:                  0.454
Method:                 Least Squares   F-statistic:                     237.8
Date:                Tue, 30 Oct 2018   Prob (F-statistic):         1.33e-184
Time:                        18:54:01   Log-Likelihood:                -9433.1
No. Observations:                1424   AIC:                         1.888e+04
Df Residuals:                    1418   BIC:                         1.891e+04
Df Model:                           5
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept        -80.3870     30.036     -2.676      0.008    -139.307     -21.467
runtime            0.6023      0.303      1.990      0.047       0.009       1.196
budget             3.0173      0.102     29.680      0.000       2.818       3.217
critics_pick      39.8173     14.667      2.715      0.007      11.046      68.589
cast_score         5.9713      5.821      1.026      0.305      -5.447      17.390
director_score    37.7703     25.515      1.480      0.139     -12.280      87.821
==============================================================================
Omnibus:                     1175.028   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            52453.970
Skew:                           3.516   Prob(JB):                         0.00
```

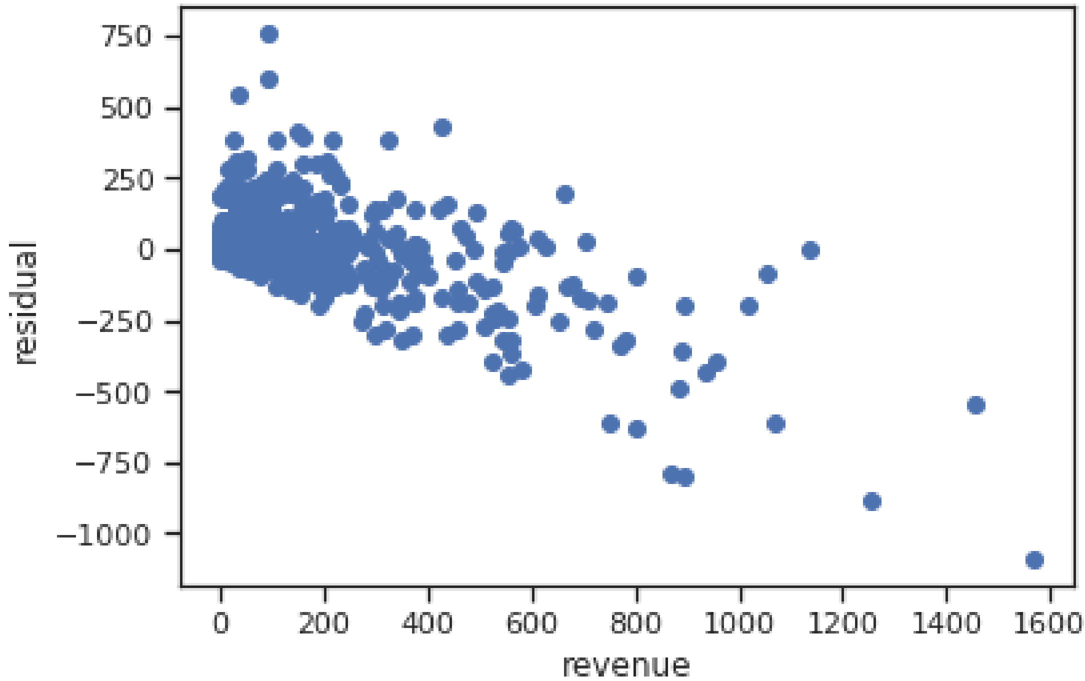
```
Kurtosis:                      31.890   Cond. No.                         816.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
317 0.5188216039279869
```



In [12]:
```
formula1 = '''revenue ~ critics_pick + budget + cast_score + director_score'''
formula2 = '''revenue ~ budget + runtime + cast_score + director_score'''

formula3 = '''revenue ~
                    budget + critics_pick + cast_score + director_score
                    + budget * critics_pick + budget * cast_score + budget * director_
                    + critics_pick * cast_score + critics_pick * director_score
                    + cast_score * director_score
                    + np.power(budget,2) + np.power(critics_pick,2) + np.power(cast_s
                    '''

results = smf.ols(formula2, data=ols_train).fit()
evaluate_ols_results(results)
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:                revenue   R-squared:                       0.453
Model:                            OLS   Adj. R-squared:                  0.452
Method:                 Least Squares   F-statistic:                     294.1
Date:                Tue, 30 Oct 2018   Prob (F-statistic):         2.84e-184
Time:                        18:54:01   Log-Likelihood:                -9436.8
No. Observations:                1424   AIC:                         1.888e+04
Df Residuals:                    1419   BIC:                         1.891e+04
Df Model:                           4
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept        -88.3987     29.958     -2.951      0.003    -147.165     -29.632
budget             2.9942      0.102     29.491      0.000       2.795       3.193
runtime            0.7232      0.300      2.410      0.016       0.134       1.312
cast_score         6.6842      5.828      1.147      0.252      -4.748      18.116
director_score    48.1002     25.286      1.902      0.057      -1.502      97.702
==============================================================================
Omnibus:                     1186.339   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            54936.454
Skew:                           3.556   Prob(JB):                         0.00
Kurtosis:                      32.586   Cond. No.                         815.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
316 0.5171849427168577
```




```
In [13]: formula = '''revenue ~
                    budget + runtime + critics_pick
                    + is_action + is_adventure+is_fantasy+is_science_fiction + is_cri
                    + cast_score + director_score

                    + is_action * cast_score
                    + is_science_fiction * cast_score
                    + is_crime * cast_score
                    + is_drama * cast_score
                    + is_thriller * cast_score
                    + is_western * cast_score
                    + is_comedy * cast_score
                    + is_romance * cast_score
                    + is_horror * cast_score
                    + is_mystery * cast_score
                    + is_history * cast_score
                    + is_war * cast_score
                    + is_music * cast_score
                    + is_documentary * cast_score
                    + is_adventure * cast_score
                    + is_fantasy * cast_score
                    + is_animation * cast_score

                    + budget * cast_score
```

```
+ budget * director_score
+ budget * runtime
+ budget * critics_pick
+ is_action * budget
+ is_science_fiction * budget
+ is_crime * budget
+ is_drama * budget
+ is_thriller * budget
+ is_western * budget
+ is_comedy * budget
+ is_romance * budget
+ is_horror * budget
+ is_mystery * budget
+ is_history * budget
+ is_war * budget
+ is_music * budget
+ is_documentary * budget
+ is_adventure * budget
+ is_fantasy * budget
+ is_animation * budget

+ director_score * cast_score
+ director_score * runtime
+ budget * director_score
+ director_score * critics_pick
+ is_action * director_score
+ is_science_fiction * director_score
+ is_crime * director_score
+ is_drama * director_score
+ is_thriller * director_score
+ is_western * director_score
+ is_comedy * director_score
+ is_romance * director_score
+ is_horror * director_score
+ is_mystery * director_score
+ is_history * director_score
+ is_war * director_score
+ is_music * director_score
+ is_documentary * director_score
+ is_adventure * director_score
+ is_fantasy * director_score
+ is_animation * director_score

+ critics_pick * cast_score
+ critics_pick * runtime
+ budget * critics_pick
+ director_score * critics_pick
+ is_action * critics_pick
```

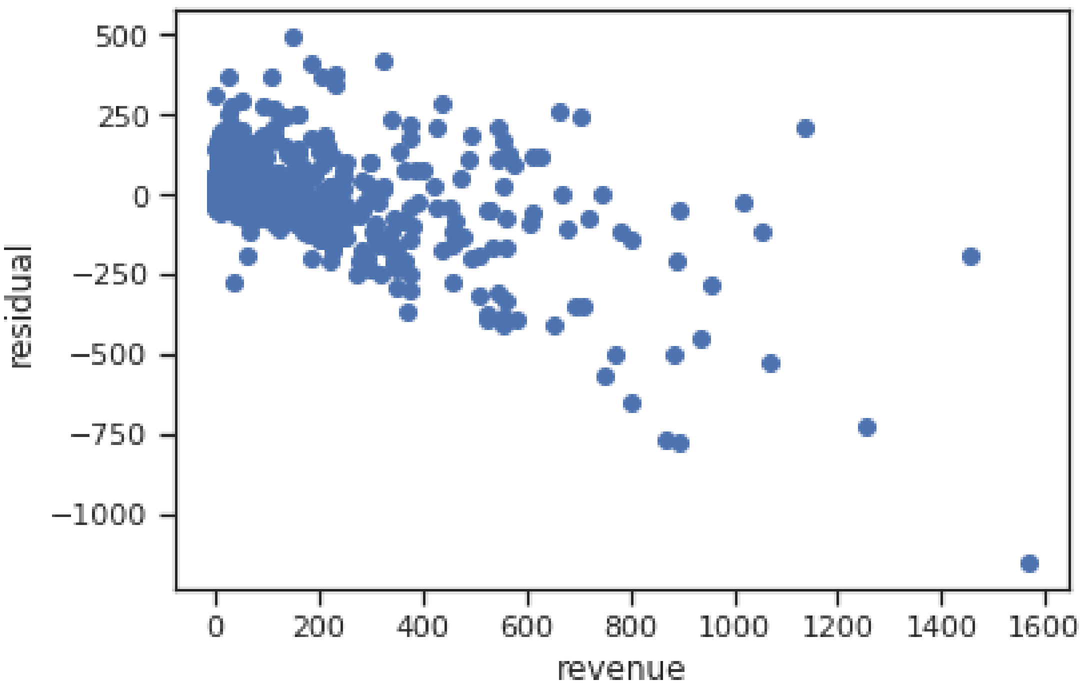
```
                           + is_science_fiction * critics_pick
                           + is_crime * critics_pick
                           + is_drama * critics_pick
                           + is_thriller * critics_pick
                           + is_western * critics_pick
                           + is_comedy * critics_pick
                           + is_romance * critics_pick
                           + is_horror * critics_pick
                           + is_mystery * critics_pick
                           + is_history * critics_pick
                           + is_war * critics_pick
                           + is_music * critics_pick
                           + is_documentary * critics_pick
                           + is_adventure * critics_pick
                           + is_fantasy * critics_pick
                           + is_animation * critics_pick

                           + np.power(budget,2)
                           + np.power(cast_score,2)
                           + np.power(director_score,2)
                           '''
    results = smf.ols(formula, data=main_df).fit()
    evaluate_ols_results(results)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                revenue   R-squared:                       0.588
Model:                            OLS   Adj. R-squared:                  0.566
Method:                 Least Squares   F-statistic:                     26.99
Date:                Tue, 30 Oct 2018   Prob (F-statistic):          1.56e-298
Time:                        18:54:01   Log-Likelihood:                -13128.
No. Observations:                2035   AIC:                         2.646e+04
Df Residuals:                    1932   BIC:                         2.704e+04
Df Model:                         102
Covariance Type:            nonrobust
===============================================================================================
                                    coef    std err          t      P>|t|      [0.025
-----------------------------------------------------------------------------------------------
Intercept                        60.3640     46.526      1.297      0.195     -30.882
budget                           -1.9696      0.562     -3.506      0.000      -3.071
runtime                          -0.2623      0.429     -0.612      0.541      -1.103
critics_pick                   -170.3003     84.467     -2.016      0.044    -335.956
is_action                       -18.0999     19.438     -0.931      0.352     -56.222
is_adventure                      1.6777     23.618      0.071      0.943     -44.642
is_fantasy                        8.4038     26.662      0.315      0.753     -43.885
is_science_fiction              -58.3573     22.267     -2.621      0.009    -102.028
```

is_crime	7.2507	21.108	0.343	0.731	-34.147
is_drama	-3.3134	16.868	-0.196	0.844	-36.395
is_thriller	-17.2467	18.507	-0.932	0.352	-53.543
is_animation	93.4999	40.225	2.324	0.020	14.611
is_family	23.7758	15.465	1.537	0.124	-6.554
is_western	22.4859	111.045	0.202	0.840	-195.295
is_comedy	11.3502	17.249	0.658	0.511	-22.478
is_romance	-26.0031	18.864	-1.378	0.168	-62.999
is_horror	22.9956	23.165	0.993	0.321	-22.435
is_mystery	53.8050	26.273	2.048	0.041	2.279
is_history	-16.3735	67.379	-0.243	0.808	-148.516
is_war	-0.8505	58.124	-0.015	0.988	-114.843
is_music	36.2912	35.503	1.022	0.307	-33.337
is_documentary	-31.1909	46.733	-0.667	0.505	-122.844
cast_score	21.0077	16.311	1.288	0.198	-10.981
director_score	157.5381	145.212	1.085	0.278	-127.252
is_action:cast_score	-4.7050	13.550	-0.347	0.728	-31.280
is_science_fiction:cast_score	-7.8328	14.889	-0.526	0.599	-37.034
is_crime:cast_score	-0.5379	13.709	-0.039	0.969	-27.423
is_drama:cast_score	-14.8932	10.738	-1.387	0.166	-35.953
is_thriller:cast_score	9.1526	11.876	0.771	0.441	-14.138
is_western:cast_score	-6.7871	64.702	-0.105	0.916	-133.681
is_comedy:cast_score	-16.8147	10.556	-1.593	0.111	-37.518
is_romance:cast_score	7.0648	11.558	0.611	0.541	-15.603
is_horror:cast_score	-33.7909	22.178	-1.524	0.128	-77.286
is_mystery:cast_score	12.9660	15.670	0.827	0.408	-17.766
is_history:cast_score	28.0615	28.136	0.997	0.319	-27.119
is_war:cast_score	-14.6295	41.263	-0.355	0.723	-95.554
is_music:cast_score	-8.8525	23.276	-0.380	0.704	-54.502
is_documentary:cast_score	2.8842	65.013	0.044	0.965	-124.619
is_adventure:cast_score	27.6523	13.320	2.076	0.038	1.529
is_fantasy:cast_score	5.8681	14.170	0.414	0.679	-21.921
is_animation:cast_score	-38.4937	18.689	-2.060	0.040	-75.146
budget:cast_score	0.3616	0.112	3.223	0.001	0.142
budget:director_score	0.7821	0.474	1.651	0.099	-0.147
budget:runtime	0.0292	0.005	5.673	0.000	0.019
budget:critics_pick	1.1472	0.306	3.755	0.000	0.548
is_action:budget	0.3963	0.239	1.659	0.097	-0.072
is_science_fiction:budget	0.7414	0.232	3.191	0.001	0.286
is_crime:budget	-0.3149	0.305	-1.034	0.301	-0.912
is_drama:budget	-0.2787	0.216	-1.290	0.197	-0.702
is_thriller:budget	0.0773	0.221	0.350	0.727	-0.357
is_western:budget	-2.0346	0.495	-4.107	0.000	-3.006
is_comedy:budget	0.2548	0.192	1.327	0.185	-0.122
is_romance:budget	0.8720	0.283	3.078	0.002	0.316
is_horror:budget	0.4538	0.447	1.015	0.310	-0.423
is_mystery:budget	-1.0935	0.365	-2.996	0.003	-1.809
is_history:budget	-1.3594	0.637	-2.132	0.033	-2.610

is_war:budget	0.0039	0.663	0.006	0.995	-1.297
is_music:budget	-0.6779	0.706	-0.961	0.337	-2.062
is_documentary:budget	1.7005	4.367	0.389	0.697	-6.864
is_adventure:budget	-0.0399	0.225	-0.177	0.859	-0.482
is_fantasy:budget	-0.0778	0.233	-0.334	0.739	-0.535
is_animation:budget	1.3997	0.335	4.174	0.000	0.742
director_score:cast_score	-30.2260	22.564	-1.340	0.181	-74.479
director_score:runtime	-0.6707	1.148	-0.584	0.559	-2.921
director_score:critics_pick	75.7468	49.363	1.534	0.125	-21.063
is_action:director_score	-55.2976	60.621	-0.912	0.362	-174.187
is_science_fiction:director_score	-1.6977	73.293	-0.023	0.982	-145.439
is_crime:director_score	-69.0939	52.491	-1.316	0.188	-172.038
is_drama:director_score	-144.9717	53.868	-2.691	0.007	-250.618
is_thriller:director_score	-4.2984	50.964	-0.084	0.933	-104.248
is_western:director_score	-14.5950	172.020	-0.085	0.932	-351.960
is_comedy:director_score	-67.6830	54.164	-1.250	0.212	-173.910
is_romance:director_score	93.6032	60.745	1.541	0.124	-25.530
is_horror:director_score	-151.3780	107.462	-1.409	0.159	-362.133
is_mystery:director_score	39.3606	60.770	0.648	0.517	-79.821
is_history:director_score	-140.4843	92.940	-1.512	0.131	-322.758
is_war:director_score	90.4781	93.607	0.967	0.334	-93.103
is_music:director_score	-30.2200	129.807	-0.233	0.816	-284.797
is_documentary:director_score	1.623e-15	6.76e-14	0.024	0.981	-1.31e-13
is_adventure:director_score	11.1382	63.828	0.175	0.861	-114.041
is_fantasy:director_score	-41.3167	69.285	-0.596	0.551	-177.198
is_animation:director_score	-424.8457	119.195	-3.564	0.000	-658.610
critics_pick:cast_score	6.4104	13.733	0.467	0.641	-20.523
critics_pick:runtime	0.8973	0.735	1.220	0.222	-0.545
is_action:critics_pick	-18.9608	40.892	-0.464	0.643	-99.157
is_science_fiction:critics_pick	137.8525	45.666	3.019	0.003	48.293
is_crime:critics_pick	29.2500	35.280	0.829	0.407	-39.941
is_drama:critics_pick	20.7161	29.802	0.695	0.487	-37.731
is_thriller:critics_pick	-30.6877	33.205	-0.924	0.356	-95.809
is_western:critics_pick	69.9145	133.014	0.526	0.599	-190.952
is_comedy:critics_pick	33.8225	27.506	1.230	0.219	-20.123
is_romance:critics_pick	16.6426	28.188	0.590	0.555	-38.640
is_horror:critics_pick	130.3221	65.125	2.001	0.046	2.600
is_mystery:critics_pick	-135.8938	45.465	-2.989	0.003	-225.059
is_history:critics_pick	30.4758	63.238	0.482	0.630	-93.547
is_war:critics_pick	26.2229	81.325	0.322	0.747	-133.271
is_music:critics_pick	4.1487	62.501	0.066	0.947	-118.428
is_documentary:critics_pick	55.0764	80.144	0.687	0.492	-102.102
is_adventure:critics_pick	-41.8335	35.610	-1.175	0.240	-111.672
is_fantasy:critics_pick	30.0092	42.796	0.701	0.483	-53.921
is_animation:critics_pick	21.7150	51.510	0.422	0.673	-79.305
np.power(budget, 2)	-0.0024	0.001	-1.834	0.067	-0.005
np.power(cast_score, 2)	-3.5869	3.940	-0.910	0.363	-11.313
np.power(director_score, 2)	117.6883	87.902	1.339	0.181	-54.704

```
==============================================================================
Omnibus:                     1110.031   Durbin-Watson:                   1.519
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            18876.062
Skew:                           2.184   Prob(JB):                         0.00
Kurtosis:                      17.266   Cond. No.                     1.23e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.3e-21. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
304 0.49754500818330605
```



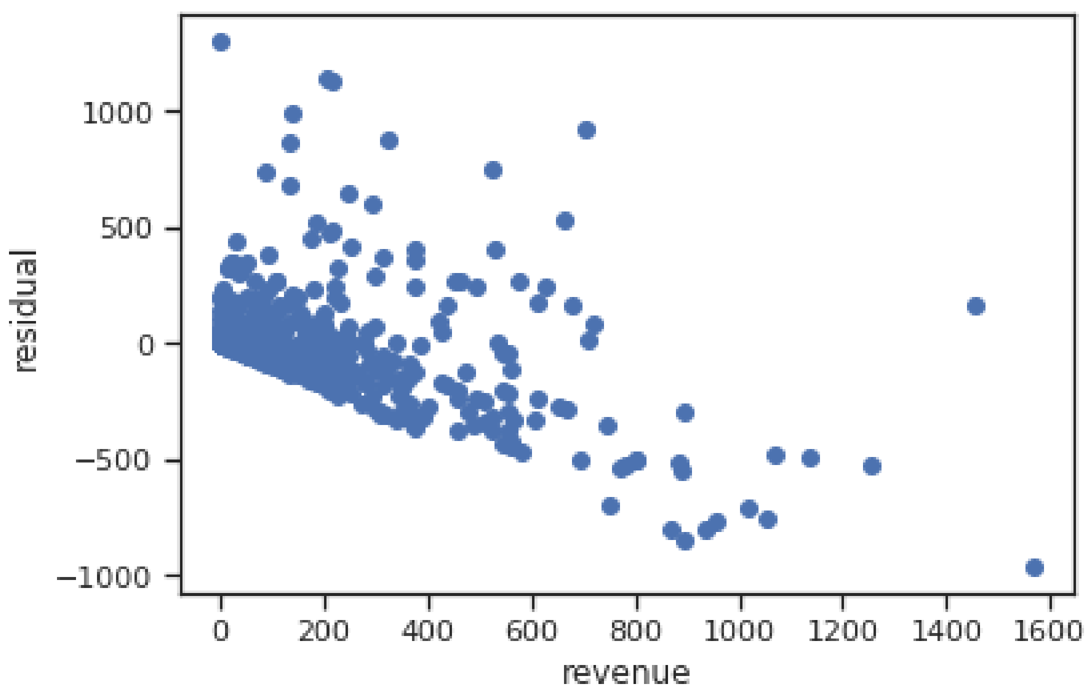
1.10.1 Non-Linear Models

```
In [14]: def evaluate_model(model,X,Y):
             if hasattr(model,'coef_'):
                 print('Intercept: ' + str(model.intercept_))
                 print('\nCoefficients: ' + str([l + ': ' + str(c) for l,c in zip(x_train.colu
                 print('\nR-Squared: ' + str(model.score(x_test,y_test)))
             print('\nResidual Plot')
             plt.scatter(Y, (model.predict(X) - Y))
             error_rate = (model.predict(X) - Y)/Y
```

```
        print('total in test set: ' +str(len(Y)))
        valid_predictions = np.sum(error_rate < limit)
        print('percent with less than {} error: '.format(limit) + str(valid_predictions))
        print('accuracy: ' + str(valid_predictions/len(Y)))
        plt.xlabel('revenue')
        plt.ylabel('residual')
        plt.show()
```

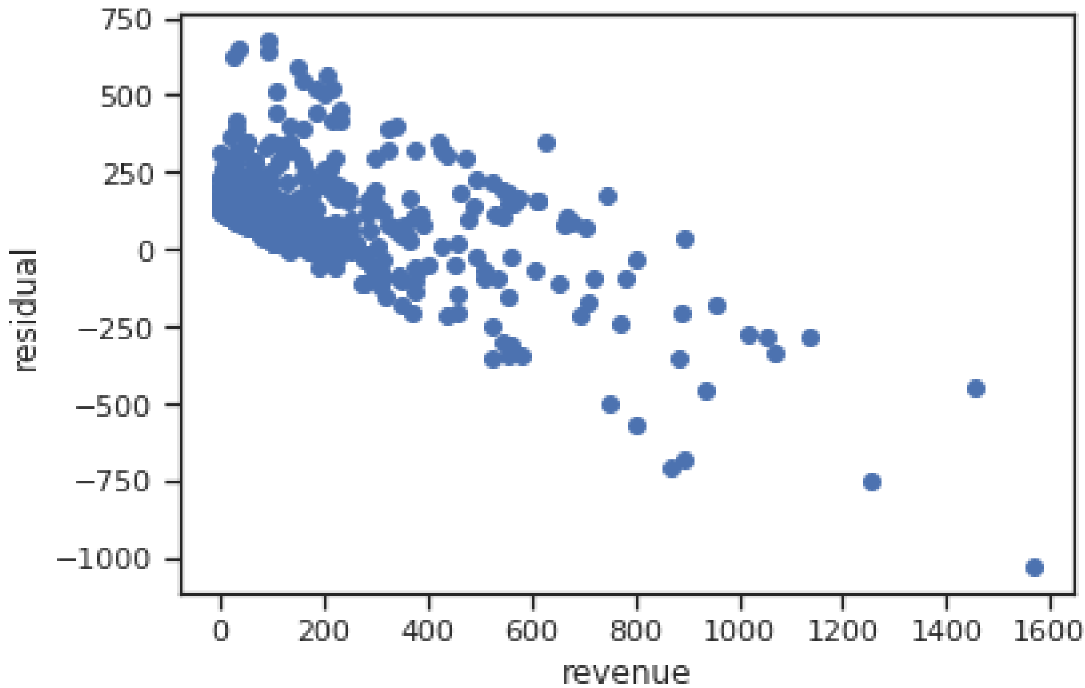
```
In [15]: tree_model = DecisionTreeRegressor()
         tree_model.fit(x_train,y_train)
         print(tree_model.score(x_test,y_test))
         evaluate_model(tree_model, x_test,y_test)
         tree.export_graphviz(tree_model, out_file='tree.dot',feature_names = x_train.columns)
```

```
-0.012608570344341485

Residual Plot
total in test set: 611
percent with less than 0.2 error: 370
accuracy: 0.6055646481178396
```



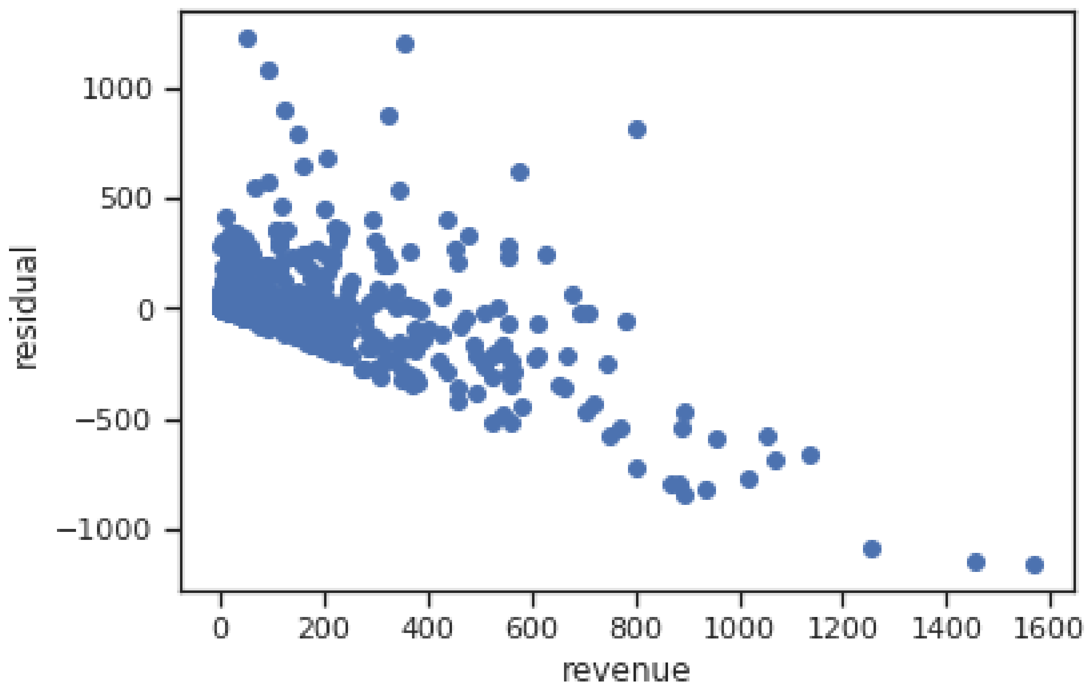

```
In [16]: ada_model = AdaBoostRegressor()
         ada_model.fit(x_train,y_train)
         print(ada_model.score(x_test,y_test))
         evaluate_model(ada_model, x_test,y_test)
```

```
0.1481071485753228

Residual Plot
total in test set: 611
percent with less than 0.2 error: 113
accuracy: 0.18494271685761046
```



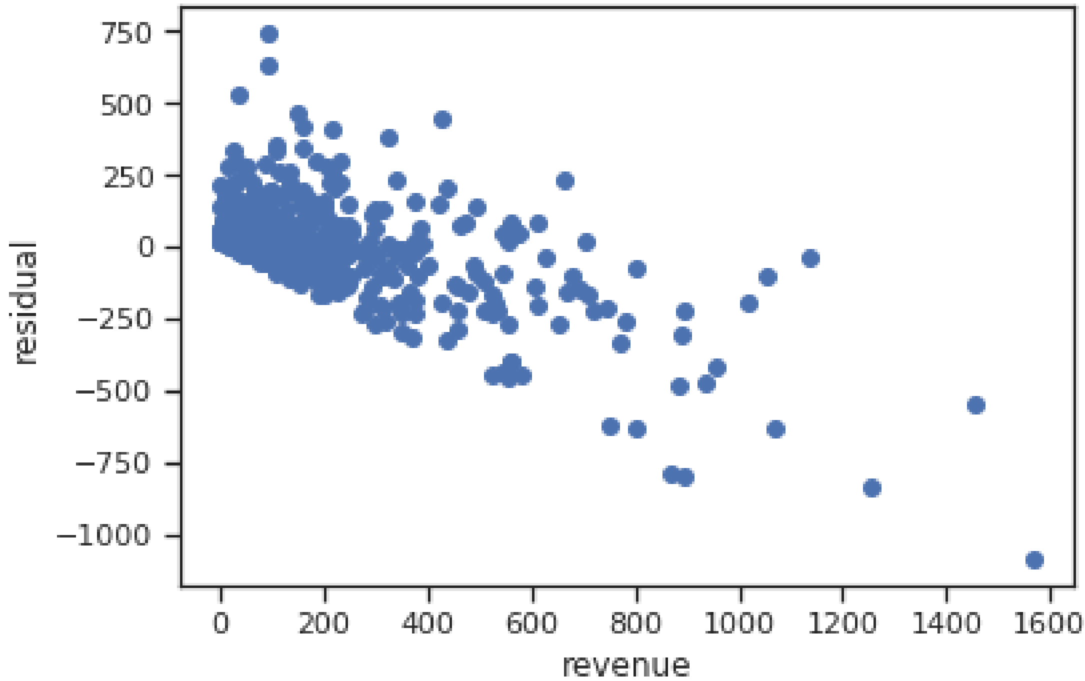

```
In [17]: knn_model = KNeighborsRegressor(n_neighbors = 1)
         knn_model.fit(x_train,y_train)
         print(knn_model.score(x_test,y_test))
         evaluate_model(knn_model, x_test,y_test)
```

```
-0.0766122375241214

Residual Plot
total in test set: 611
percent with less than 0.2 error: 357
accuracy: 0.5842880523731587
```




```
In [18]: mlp_r = MLPRegressor(hidden_layer_sizes=(5,3),max_iter = 2000)
         mlp_r.fit(x_train,y_train)
         print(mlp_r.score(x_test,y_test))
         evaluate_model(mlp_r, x_test,y_test)
```

```
0.47588013545778296

Residual Plot
total in test set: 611
percent with less than 0.2 error: 273
accuracy: 0.44680851063829785
```



1.11 Helper functions for scrapping

Functions for retrieving data from NYT movies API and OMDB API. Results are saved in csv files.

```
In [19]: '''#Codes for scraping, dont run. saved to csv file.
         NYT_API_KEY = '53223e11b006467490bde835d45b0c74'

         all_ny_df = []
         for offset in range(0,8000,20):
             url = 'http://api.nytimes.com/svc/movies/v2/reviews/search.json?opening-date=1990
             ny_json = pd.read_json(url, orient = 'records')
             ny_df = json_normalize(ny_json['results'])
             if ny_df.empty:
                 break
             all_ny_df.append(ny_df)

         ny_df = pd.concat(all_ny_df)
         print(ny_df.tail())
         ny_df.to_csv('NY Movie Reviews.csv')'''

Out[19]: "#Codes for scraping, dont run. saved to csv file.\nNYT_API_KEY = '53223e11b006467490
```

title = 't=' + nytdata['display_title'][1].replace(' ', '+') req = 'http://www.omdbapi.com/?apikey='+ OMDB_API_KEY + '&'+ title print(pd.read_json(req))

In [20]:
```
'''OMDB_API_KEY = 'd42886f4'

def fetch_omdb(title):
    title = 't=' + title.replace(' ', '+')
    print (title)
    req = 'http://www.omdbapi.com/?apikey='+ OMDB_API_KEY + '&'+ title
    omdb_df = pd.read_json(req)
    return omdb_df

count = 0
omdb_df_list = []
for title in tmdb_df['title'].tolist():
    count += 1
    omdb_df_list.append(fetch_omdb(title))
    if count > 5:
        break

complete = pd.concat(omdb_df_list,axis=0)
complete.to_csv('omdb_data.csv')
'''
```

Out[20]:
```
"OMDB_API_KEY = 'd42886f4'\n\ndef fetch_omdb(title):\n    title = 't=' + title.replac
```

In [ ]: