WHETHER A MOVIE IS WORTH THE INVESTMENT

PREDICTING GROSS EARNING OF MOVIES IN THE USA

Jhih-Rou Huang

INTRODUCTION

Motivation: What kind of movies have the potential highest gross earning and are the best choice for the investors.

Objectives and goals: This project aims to help investors decide which and what kind of movies are worth investing in. In this way, the new investors can use the result of this project as a reference to decide what kind of movie are the most worthwhile for them to invest in. Also, the film companies can know which features of the movies influence their gross earning the most and adjust their filming plan to increase the revenue.

METHODOLOGY

Tools

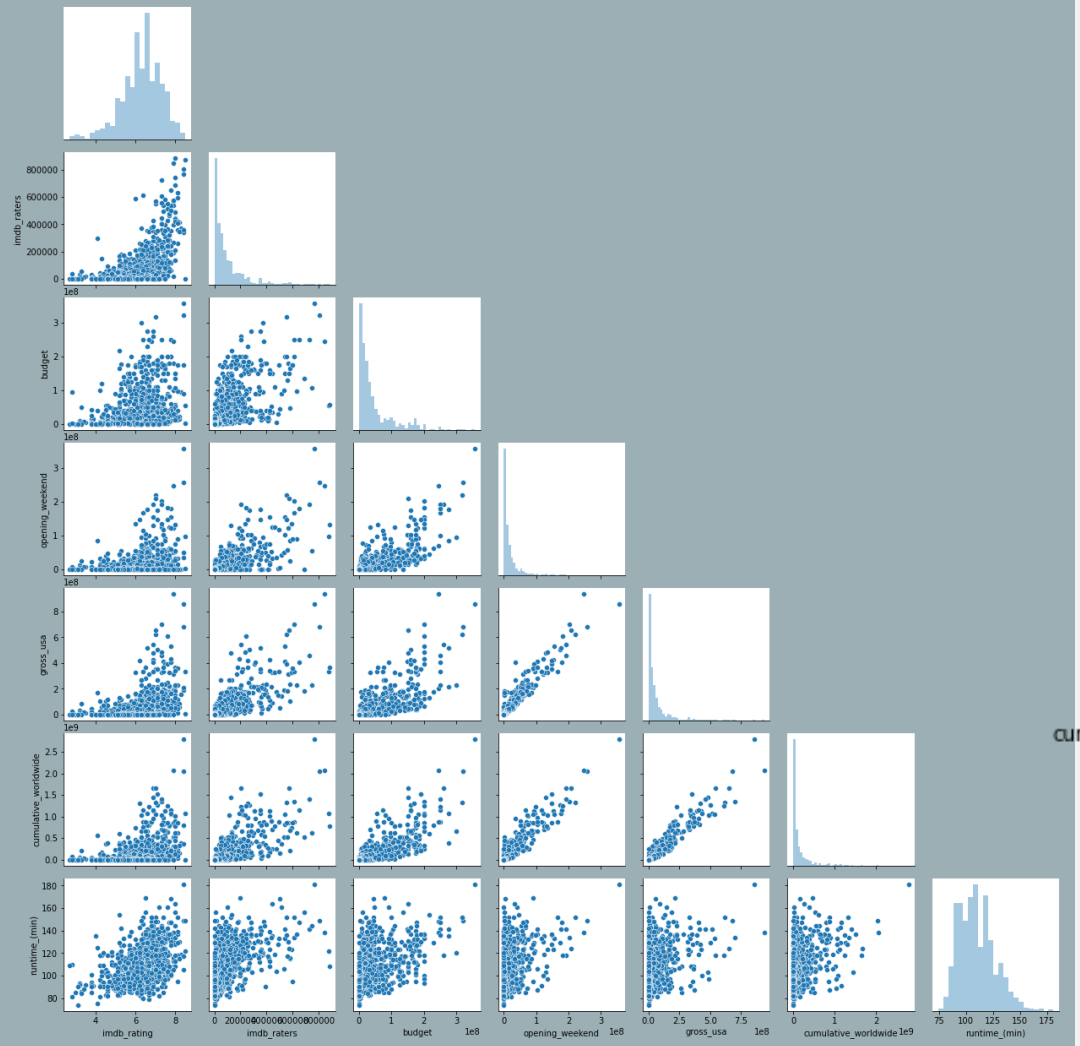
- We used beautiful soup to scrap the data from the IMDb website.
- Also, the pandas, NumPy, and sklearn are used in organizing the data and doing the regression analysis. Finally, the visualization and figures are displayed by the matplotlib and seaborn modules.

METHODOLOGY

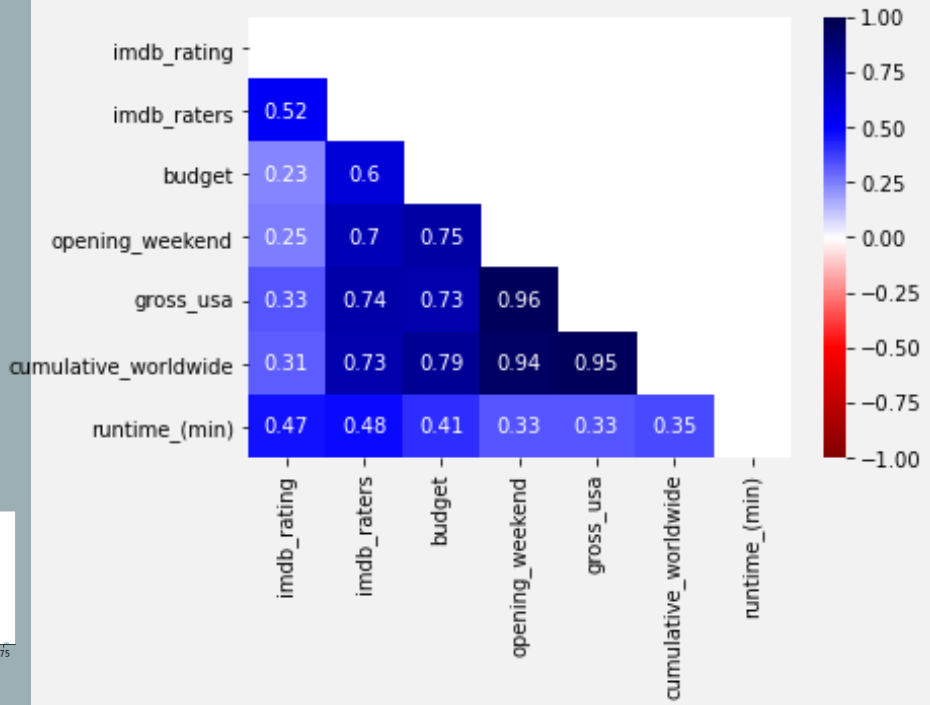
Data

- The IMDb website is the source of data in this project.
- We gathered movie data from 2015 to 2020 using BeautifulSoup.
- We scrapped multiple pages, and each page has 100 movies.
- The raw database has 7856 data, and after cleaning, 856 data points were left.
- The factors, runtime, budget, MPAA Rating, genre, director, writer, stars, production, country, language, and release date (release month and years since release) are used in the prediction model.

CORRELATION



- gross_usa 1.000000
- opening_weekend 0.956543
- cumulative_worldwide 0.954895
- imdb_raters 0.739396
- budget 0.732849
- imdb_rating 0.333552
- runtime_(min) 0.332921
- Name: gross_usa, dtype: float64



All features in the dataset:
imdb rating
imdb raters
mpaa
genres
director
writer
stars
country
language
release date
budget
opening weekend
gross usa
cumulative worldwide
production companies
runtime (min)

METHODOLOGY

Features Included in the Model

- imdb rating
- imdb raters
- mpaa
- genres
- director
- writer
- stars
- country
- language
- release date
- budget
- opening weekend
- gross usa
- cumulative worldwide
- production companies
- runtime (min)

Feature Engineering

R-square Value
Runtime: 0.1108
Budget: 0.5382

Feature: Runtime

```
X, y = movies_df_drop[['runtime (min)']], movies_df_drop['gross usa']
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.1108
```

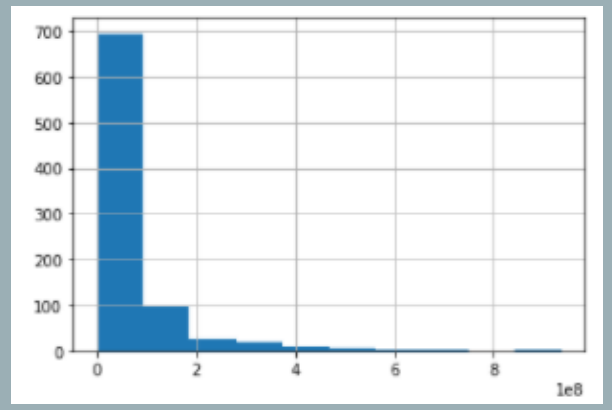
Feature: Budget

```
X, y = movies_df_drop[['budget', 'runtime (min)']], movies_df_drop['gross usa']
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

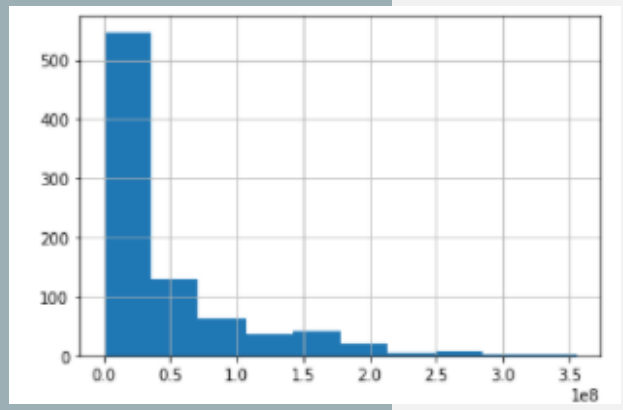
```
R-squared: 0.5382
```

Seaborn for Exploring Distributions

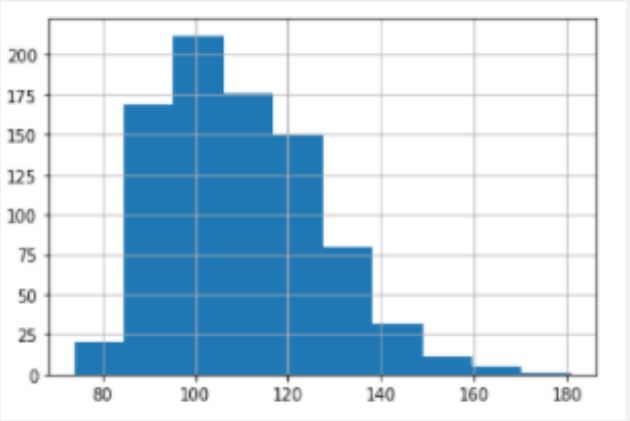
Gross USA



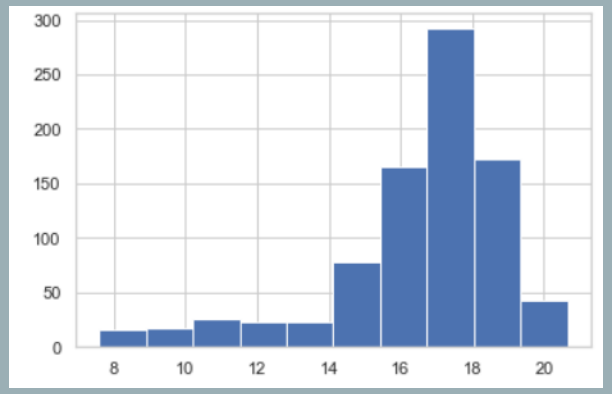
Budget



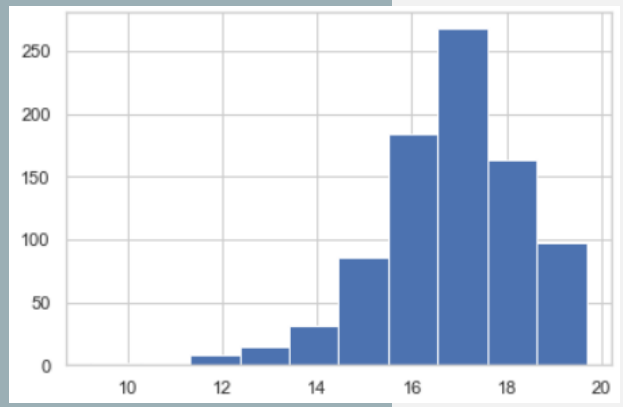
Runtime (min)



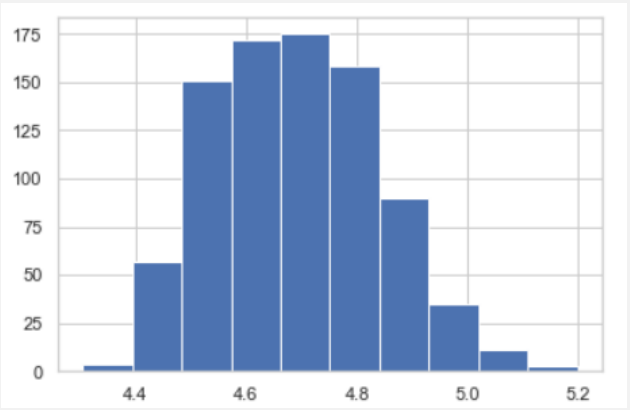
Log Gross USA



Log Budget



Log Runtime (min)



Feature Engineering

R-square Value
Runtime: 0.1108
Budget: 0.5382

Feature: Runtime

```
X, y = movies_df_drop[['runtime (min)']], movies_df_drop['gross usa']
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.1108
```

Feature: Budget

```
X, y = movies_df_drop[['budget', 'runtime (min)']], movies_df_drop['gross usa']
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.5382
```

```
X, y = movies_df_drop[['LOG_budget', 'LOG_runtime (min)']], movies_df_drop['LOG_gross usa']
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.4967
```

Try imdb rating data

```
X, y = movies_df_drop[['budget', 'runtime (min)','imdb rating']], movies_df_drop['gross usa']
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.5662
```

Adding polynomial terms

```
X1 = X.copy()
X1['runtime (min)**2'] = X1['runtime (min)'] ** 2
X1['budget**2'] = X1['budget'] ** 2
lr = LinearRegression()
lr.fit(X1, y)
print('R-squared: {:.4f}'.format(lr.score(X1, y)))
```

```
R-squared: 0.5631
```

Adding interaction terms

```
X2 = X1.copy()

# multiplicative interaction
#X3['B_x_C'] = X3['cumulative_worldwide'] * X3['budget']

# division interaction
X2['B_/_R'] = X2['budget'] / X2['runtime (min)']
lr = LinearRegression()
lr.fit(X2, y)
print('R-squared: {:.4f}'.format(lr.score(X2, y)))
```

```
R-squared: 0.5632
```

Feature Engineering

Get dummy data for the categorical data

R-square Value

Runtime: 0.1108
Budget: 0.5382
MPAA Rating: 0.5403
Genre: 0.5889
Director: 0.609 (use the top ten director)
Writer: 0.6208 (use the top ten writer)
Stars: 0.6306 (use the top ten stars)
Production company: 0.6651 (use the top ten)
Release month: 0.6698
Years since release: 0.67
Country: 0.6755 (use top ten)
Language: 0.6767

It turns out that R square will continue to go up every time we add new features. This is because the R square score does not normalize for the number of features.

Feature: MPAA Rating

```
mpaa_df = pd.get_dummies(movies_df_drop['mpaa'])
```

```
df_added_mpaa_dummies = pd.concat([movies_df_drop,mpaa_df], axis=1)
```

Feature: Release Month

```
movies_df_drop['month'] = movies_df_drop['release date'].apply(lambda x: x.strftime("%b"))
```

```
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py: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
  """Entry point for launching an IPython kernel.
```

```
month_df = pd.get_dummies(movies_df_drop['month'])
```

```
month_model = pd.concat([prod_model,month_df], axis=1)
```

```
X, y = month_model, y
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.6698
```

Feature: Years since release

```
date = pd.to_datetime(datetime.now().date())
```

```
movies_df_drop['years since release'] = movies_df_drop['release date'].apply(lambda x: (((date-pd.to_datetime(x))).days/ 365.
```

```
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\ipykernel_launcher.py: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
  """Entry point for launching an IPython kernel.
```

```
month_model['years since release'] = movies_df_drop['years since release']
```

```
X, y = month_model, y
lr = LinearRegression()
lr.fit(X, y)
print('R-squared: {:.4f}'.format(lr.score(X, y)))
```

```
R-squared: 0.6700
```

Cross Validation Train/ Validation/Test

The model with 12 features.

- Runtime
- Budget
- MPAA Rating
- Genre
- Director
- Writer
- Stars
- Production company
- Release month
- Years since release
- Country
- Language:

The generalization error between the train and validation data set:

No overfitting: r square decrease< 0.005

```
y=movies_df_drop['gross usa']
X, y = df_lan_model, y
```

```
X, X_test, y, y_test = train_test_split(X, y,test_size=.2,random_state=42)

X_train, X_val, y_train, y_val = train_test_split(X, y,test_size=.20,random_state=42)
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
lr = model.fit(X_train,y_train)

r_squared_train = lr.score(X_train, y_train)
r_squared_val = lr.score(X_val, y_val)
r_squared_test = lr.score(X_test, y_test)

print(r_squared_train)
print(r_squared_val)
print(r_squared_test)
```

```
0.6526702703984493
0.6511567706735606
0.6024749770630412
```

POLYNOMIAL REGRESSION

- The second degree polynonmial regression does not perform as good as the linear regression

```
# Let's first test Linear Regression vs Polynomial Features
lr = LinearRegression()

poly = PolynomialFeatures(degree=2)

X_train_poly = poly.fit_transform(X_train.values)
X_val_poly = poly.transform(X_val.values)
X_test_poly = poly.transform(X_test.values)

lr_poly = LinearRegression()
```

```
lr.fit(X_train, y_train)
print(f'Linear Regression val R-squared: {lr.score(X_val, y_val):.3f}')

lr_poly.fit(X_train_poly, y_train)
print(f'Degree 2 polynomial regression val R-squared: {lr_poly.score(X_val_poly, y_val):.3f}')
```

```
Linear Regression val R-squared: 0.651
Degree 2 polynomial regression val R-squared: 0.044
```

PERFORM 5-FOLD CROSS VALIDATION ON THE DATA. BEFORE DOING SO, LEAVE 20% OF THE DATA AS TEST. USE 42 FOR YOUR RANDOM STATE VALUE.

- Slightly overfit (r square decrease of .01-.05)

```
# Here we'll validate our Linear model vs. the Ridge model
kf = KFold(n_splits=5, shuffle=True, random_state = 61)
cv_lr_r2s, cv_lr_reg_r2s = [], []

for train_ind, val_ind in kf.split(X,y):

    X_train, y_train = X[train_ind], y[train_ind]
    X_val, y_val = X[val_ind], y[val_ind]

    #Linear Regression
    lr = LinearRegression()
    #lr_reg = Ridge(alpha=1)

    lr.fit(X_train, y_train)
    cv_lr_r2s.append(lr.score(X_val, y_val))

print('Simple regression scores: ', cv_lr_r2s)

print(f'Simple mean cv R-squared: {np.mean(cv_lr_r2s):.3f} +- {np.std(cv_lr_r2s):.3f}')
```

```
Simple regression scores:  [0.2200456733757784, 0.39982791519090766, 0.11796112756354382, 0.47177737201976844, 0.6363361070193357]
Simple mean cv R-squared: 0.369 +- 0.183
```

Fit the final model on the training data then score it on the test data set. Report the `train_r2` and `test_r2`.

```
X, y = df_lan_model, y
lm = LinearRegression()
lm.fit(X_train, y_train)
train_score = lm.score(X_train, y_train)
test_score = lm.score(X_test, y_test)
print('Train Score: ', lm.score(X_train, y_train))
print('Test Score: ', lm.score(X_test, y_test))
list(zip(X.columns, lr.coef_))
```

```
Train Score:  0.653864551953895
Test Score:  0.615873986437228
```

REGULARIZATION

- Standardize your variables beforehand.
- Set the value of alpha to 2000.
- Two coefficients =0

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
## .fit finds the mean and standard deviation of each variable in the training set

y=movies_df_drop['gross usa']
X, y = df_lan_model, y

std = StandardScaler()
std.fit(X_train)

## apply the scaler to the train set
X_tr = std.transform(X_train)
## Apply the scaler to the test set
X_te = std.transform(X_test)

from sklearn.linear_model import Ridge

lr_model_ridge = Ridge(alpha = 2000)
lr_model_ridge.fit(X_tr, y_train)
ridge_test_r2 = lr_model_ridge.score(X_te, y_test)
ridge_test_r2
```

```
0.38231914361264974
```

```
print(lr_model_ridge.score(X_tr, y_train))
print(lr_model_ridge.score(X_te, y_test))
list(zip(X.columns, lr_model_ridge.coef_))
```

```
0.37368216720103475
0.38231914361264974
```

REGULARIZATION

- Find the optimal penalty term through cross validation.
- Search 200 possible values.
- Five coefficients =0

```
alphas = 10**np.linspace(-2,2,200)
ridge_model = RidgeCV(alphas = alphas, cv=5)
ridge_model.fit(X_tr, y_train)
ridge_model.score(X_tr, y_train)
r_squared_test = ridge_model.score(X_te, y_test)
list(zip(X_train, lr_model_ridge.coef_))
alpha = ridge_model.alpha_
```

```
ridge_model.score(X_tr, y_train)
```

```
0.623339312783912
```

```
r_squared_test
```

```
0.6188432663352127
```

```
alpha
```

```
100.0
```

LASSO REGRESSION

- Holdout 20% of the data as a test dataset.
- Set the random state to 42.
- Standardize your features for modeling.
- Use a lambda value of 1000000 (which is VERY strong).
- Variables were dropped to zero= 35

```
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
## .fit finds the mean and standard deviation of each variable in the training set
std = StandardScaler()
std.fit(X_train)
## apply the scaler to the train set
X_tr = std.transform(X_train)
## Apply the scaler to the test set
X_te = std.transform(X_test)
lasso = Lasso(alpha = 1000000)
lasso.fit(X_tr, y_train)
test_r_squared = lasso.score(X_te, y_test)
print(test_r_squared)
print(list(zip(X.columns, lasso.coef_)))
variables_dropped_count = 5
```

```
0.6585703737457378
[('budget', 6519441 3.32220786), ('runtime (min)', 9353183.076599974), ('G', -974594.5081533596), ('Not Rated', -867616.43581
80729), ('PG', 807691.2562040921), ('PG-13', 0.0), ('R', -1874829.9427205333), ('TV-MA', -0.0), ('Unrated', -12327.707368926
694), ('Action', -4581098.352883359), ('Adventure', -3586348.149411958), ('Animation', 5261898.389610467), ('Biography', -82
5230.2780059004), ('Comedy', 0.0), ('Crime', -0.0), ('Drama', -2853914.997595587), ('Family', 879072.5668273158), ('Fantas
y', -10411443.50272233), ('History', -846673.1899407103), ('Horror', 4555806.301511562), ('Music', 1126851.170624407), ('Mus
ical', 5560736.750434426), ('Mystery', -0.0), ('Romance', 233173.7060148139), ('Sci-Fi', 6004727.9929545475), ('Sport', -310
1997.8207184756), ('Thriller', -4087099.266983791), ('War', 0.0), ('Western', -1072894.2947188336), ('Antoine Fuqua', 0.0),
('Clint Eastwood', -150262.08050368633), ('David F. Sandberg', 786788.3075790554), ('Deon Taylor', 0.0), ('Guy Ritchie', -20
00129.577419296), ('M. Night Shyamalan', 1560845.8593141716), ('Paul Feig', 0.0), ('Ron Howard', -7280488.797675501), ('Stev
en Spielberg', -4542648.808577184), ('Tyler Perry', 0.0), ('Tyler Perry', 0.0), ('Leigh Whannell', 0.0), ('Christina Hodso
n', -7576.656466760832), ('Rhett Reese', 11884855.083542204), ('Taylor Sheridan', 211029.98408833443), ('Jon Lucas', 810049.
5244399923), ('Nicholas Stoller', -182549.6530014529), ('Chris McKenna', 8193306.408218078), ('Max Landis', -1922038.9746282
822), ('Jordan Peele', 5260831.040321721), ('Dwayne Johnson', 0.0), ('Kevin Hart', 5531019.098670346), ('Matthew McConaughe
y', 0.0), ('Mark Wahlberg', -0.0), ('Woody Harrelson', -2511070.775836245), ('Margot Robbie', -0.0), ('Anna Kendrick', 17535
29.232526865), ('Tom Hanks', 3987112.1298997086), ('James McAvoy', -0.0), ('Michael Fassbender', -6870452.787451604), ('Colu
mbia Pictures', 2352985.2209361256), ('Universal Pictures', 11112931.033647658), ('Warner Bros.', -2409342.810738633), ('Wal
t Disney Pictures', 14213754.749552216), ('Paramount Pictures', -5270372.898096053), ('Blumhouse Productions', 686673.155292
1026), ('Twentieth Century Fox', -1563419.9909731757), ('New Line Cinema', 3466191.3267109897), ('Perfect World Pictures', -
3499956.0378563534), ('Summit Entertainment', -2183871.1155222207), ('Apr', 905577.8472696188), ('Aug', -0.0), ('Dec', 20923
01.860201295), ('Feb', 0.0), ('Jan', -1871843.999719024), ('Jul', 467798.6838514635), ('Jun', 1112593.9612148337), ('Mar',
0.0), ('May', -73331.40023846933), ('Nov', -0.0), ('Oct', -0.0), ('Sep', -0.0), ('years since release', 0.0), ('USA', 417888
2.7007745025), ('UK', -0.0), ('China', -2564086.4901862387), ('Canada', 0.0), ('France', -2355134.3832212733), ('Hong Kong',
0.0), ('Germany', 173233.9014943932), ('Australia', 514105.38897836563), ('Ireland', -0.0), ('Japan', 1390251.5505561929),
('American Sign Language', -0.0), ('Arabic', 0.0), ('English', 0.0), ('Hebrew', 0.0), ('Hindi', -0.0), ('Mandarin', 0.0),
('Portuguese', -2530555.7132914835), ('Tswana', -0.0)]
```

LASSO sets many variables to 0 (with a high enough alpha parameter). This is its feature selection property. To help remember that LASSO does this and not ridge, look no further than the name: The shrinkage part means making the coefficients smaller by penalizing their size in the cost function, and the selection part is zeroing out coefficients.

LASSO

- Standardize your features
- Use Cross validation to find the optimal lambda

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np
alphas = 10**np.linspace(-2,2,200)
lasso_model = LassoCV(alphas = alphas, cv=5)
lasso_model.fit(X_tr, y_train)
r_squared_train = lasso_model.score(X_tr, y_train)
r_squared_test = lasso_model.score(X_te, y_test)
alpha = lasso_model.alpha_

print(r_squared_train)
print(r_squared_test)
print(alpha)
```

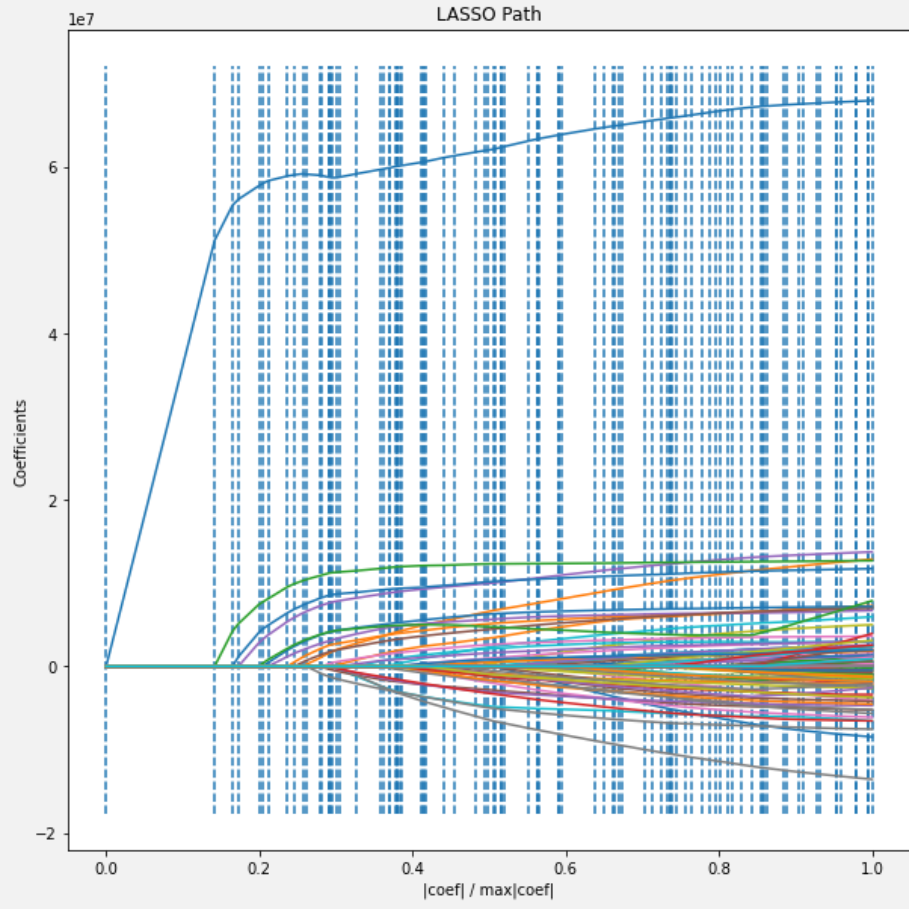
```
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
23929886976e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
13555889664e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
0286581939e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:531:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 2.22441
05078643123e+17, tolerance: 700459412634788.9
  positive)
```

```
0.6503682533669355
0.6395931866406983
0.01
```

Using LassoCV to find the best alpha via Cross-Validation
In the previous section, we found the best alpha value by comparing the performance on a single validation set. An even better, though more computationally intensive method, is to do a full cross-validation when comparing the different alphas. Fortunately, the `LassoCV` in sklearn handles this "under the hood". You pass the `LassoCV` the list of alphas and the number of folds to use for Cross-Validation.

LARS PATH

- Plot the LARS path of the lasso regression.
- The LARS path indicates the values of coefficients at a given regularization strength.
- The vertical lines indicate where a new variable 'enters' the model, becoming a non-zero coefficient. Variables that enter the model earliest are the most essential features.
- The far left of the chart indicates a penalty so high that all variables are zero, while the far right indicates a traditional regression.



COMPARISON

Lasso zeroed out most of the coefficients and dropped the noisy collinear clone, performing feature selection to keep the features we really wanted.

LASSO

- Pro: great for trimming features and focusing interpretation on a few key ones
- Con: risk of discarding features that are actually useful

Ridge:

- Pro: great for smoothly handling multicollinearity, very nice when working with sparse features
- Con (ish): will never fully discard features

As always, you have to validate to choose between the two. If the mapping from features to target truly depends on only a few key features, LASSO should outperform. If instead the target actually depends on many features (even if only a little dependent), Ridge should work better.

ELASTIC NET REGRESSION

- Holdout 20% of the data as a test dataset.
- Set the random state to 42.
- Standardize your features for modeling.
- Use a value alpha = 1.0 and an l1_ratio of 0.5 for the regression model.
- Variables are dropped to zero =6

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)

## .fit finds the mean and standard deviation of each variable in the training set
std = StandardScaler()
std.fit(X_train.values)

## apply the scaler to the train set
X_tr = std.transform(X_train.values)

## Apply the scaler to the test set
X_te = std.transform(X_test.values)

enet = ElasticNet(alpha = 1, l1_ratio=.5)
enet.fit(X_tr, y_train)
test_r_squared = enet.score(X_te, y_test)

print(test_r_squared)
print(list(zip(X_train.columns, enet.coef_)))

enet = ElasticNet(alpha = 10000, l1_ratio=1)
enet.fit(X_tr, y_train)

#list(zip(X_train.columns, enet.coef_))

variables_dropped_count = 3
```

```
0.6235638244354966
[('budget', 30920152.718680665), ('runtime (min)', 14117837.557477562), ('G', 1854177.3118154828), ('Not Rated', -2056957.3468167402), ('PG', 730715.7609687203), ('PG-13', 2882894.401284864), ('R', -2810231.7450352088), ('TV-MA', -988010.5987230444), ('Unrated', -1117047.8915225654), ('Action', 802733.7907952385), ('Adventure', 4203144.529439556), ('Animation', 4620161.990595711), ('Biography', -2491877.8887995933), ('Comedy', -788685.4433918414), ('Crime', -1007145.9592850537), ('Drama', -6713351.902511047), ('Family', 2098574.78083172), ('Fantasy', -4038296.6522460724), ('History', -1978395.4862970933), ('Horror', 727820.5626588205), ('Music', 1749057.362621418), ('Musical', 4997869.427744919), ('Mystery', -1139483.2104545657), ('Romance', -1270193.834819121), ('Sci-Fi', 7980901.925251486), ('Sport', -3379609.6834957), ('Thriller', -4003727.8664741893), ('War', -100303.84195900767), ('Western', -1621028.273167482), ('Clint Eastwood', 724291.8730400357), ('Deon Taylor', -310343.2707975644), ('Guy Ritchie', -1778616.315330062), ('Jonathan Levine', -138319.01263693263), ('Malcolm D. Lee', -497590.47832939646), ('Marielle Heller', 744825.0420820967), ('Mike Mitchell', -376061.0845549163), ('Paul Feig', 689905.3917540942), ('Steven Spielberg', -2674220.825821815), ('Tyler Perry', 275188.83169750654), ('Tyler Perry', 275188.83159661083), ('Jon Lucas', 1143365.6873926038), ('Taylor Sheridan', 394969.77478107344), ('Christina Hodson', -453968.66150679125), ('Leigh Whannell', -368507.9690052241), ('Rhett Reese', 8394885.459059948), ('Chris Weitz', 5187915.161642276), ('Jonathan Aibel', 1539547.7346976558), ('Lee Hall', -445592.4272329467), ('W. Bruce Cameron', 1091284.1102650147), ('Dwayne Johnson', 2636121.556266583), ('Kevin Hart', 5476298.613986454), ('Matthew McConaughey', 124649.94947725686), ('Mark Wahlberg', 192656.2555367722), ('Margot Robbie', 484650.59848335), ('Woody Harrelson', -2149227.848732093), ('Nicole Kidman', -705439.1751894793), ('Samuel L. Jackson', 2765928.9375837035), ('Michael Fassbender', -3887305.4195833816), ('James McAvoy', -407591.59782374697), ('Columbia Pictures', 2048187.1423112662), ('Universal Pictures', 8470995.194786709), ('Warner Bros.', -1358594.6975725435), ('Walt Disney Pictures', 14608840.815331588), ('Paramount Pictures', -2658240.124324549), ('Blumhouse Productions', 1511278.6804644766), ('Twentieth Century Fox', -795606.5983143478), ('New Line Cinema', 2966095.3621186675), ('Perfect World Pictures', -1686282.887158562), ('Summit Entertainment', -2780880.416174398), ('Apr', 1036779.181054047), ('Aug', -2290833.3559505153), ('Dec', 2126286.8906647405), ('Feb', 193050.8434164062), ('Jan', -3031704.005944681), ('Jul', 2405462.487022427), ('Jun', 2128645.225444909), ('Mar', 303240.9564489738), ('May', -789356.8733826525), ('Nov', 971225.3668705019), ('Oct', -1990539.8534154124), ('Sep', -618191.0059123725), ('years since release', -283612.0741179829), ('USA', 4009944.2688231207), ('UK', -374949.5921411173), ('China', -2541234.4566720864), ('Canada', 348977.26152236044), ('Hong Kong', -1009112.1722857255), ('France', -1827583.9313090127), ('Germany', 441217.9744280979), ('Australia', 1032191.7994490163), ('Japan', 1976498.764914902), ('Czech Republic', -869045.5255605456), ('American Sign Language', -1033151.220440966), ('Arabic', 808936.7131104679), ('English', 376942.75707372726), ('Hebrew', 572526.8298693211), ('Hindi', -428747.78088467714), ('Mandarin', 0.0), ('Portuguese', -1030505.0513106459), ('Tswana', 191006.4999012085)]
```

ELASTIC NET REGRESSION

- Using cross validation, search for an optimal l1_ratio and alpha value for Elastic Net regression models on 5 folds.
- Consider ElasticNetCV, and set the random state to 42.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np

alphas = 10 ** np.linspace(-2, 2, 200)

enet_model = ElasticNetCV(l1_ratio = [0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1],
                          alphas = alphas,
                          cv=5,
                          random_state=42)

enet_model.fit(X_tr, y_train)

r_squared_train = enet_model.score(X_tr, y_train)
r_squared_test = enet_model.score(X_te, y_test)
alpha = enet_model.alpha_
l1_ratio = enet_model.l1_ratio_

print("r_squared_train: ", round(r_squared_train, 3))
print("r_squared_test: ", round(r_squared_test, 3))
print("alpha: ", round(alpha, 3))
print("l1_ratio: ", l1_ratio)
```

```
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
33379803136e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
23929886976e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
13555889664e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
c:\users\15107\appdata\local\programs\python\python37\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527:
ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 1.80163
0286581939e+16, tolerance: 454407284607655.06
  tol, rng, random, positive)
```

```
r_squared_train:  0.634
r_squared_test:  0.646
alpha:  2.967
l1_ratio:  0.95
```

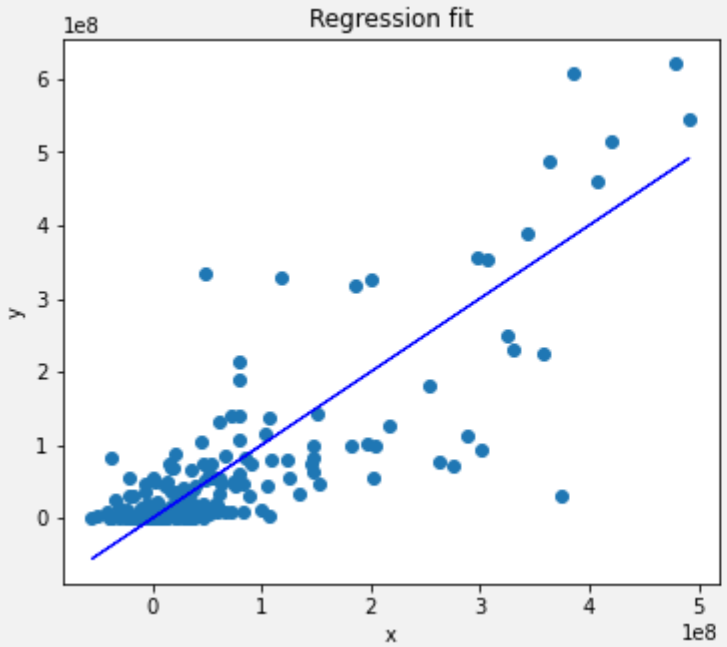
COMPARISON

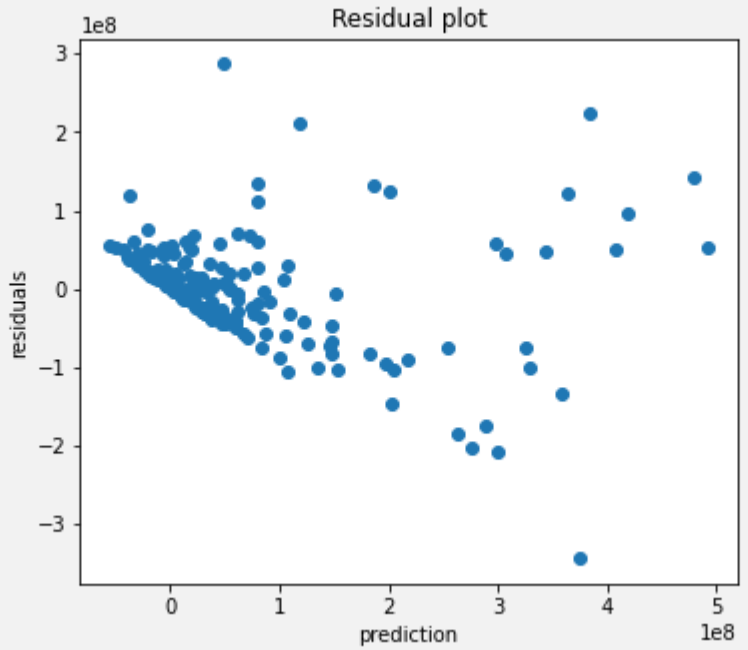
R-square

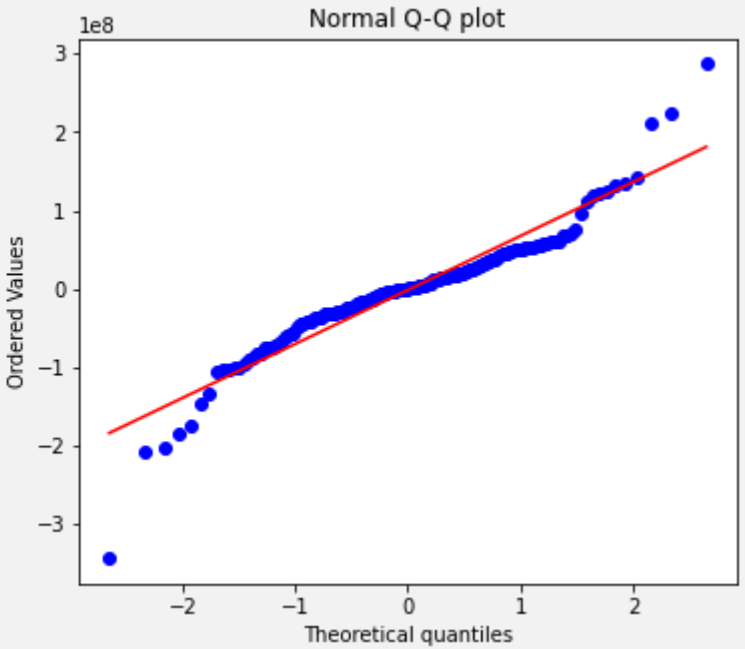
- Linear regression: 0.651
- Polynomial regression:0.044 (2 degree)
- Regularization: 0.374 (alpha=2000)
- Regularization: 0.623 (optimal alpha=100)
- Lasso: 0.659 (lambda=1000000,Variables were dropped to zero= 35)
- Lasso: 0.65 (optimal lambda= 0.01)
- Elastic net regression: 0.623 (alpha= 10000)
- Elastic net regression: 0.634 (optimal l1 ratio= 0.95 and optimal alpha= 2.967)
- Random forest regression: 0.316
- Gradient boosted regression: 0.507

DIAGNOSTIC PLOT

- Lasso: 0.65 (optimal lambda= 0.01)



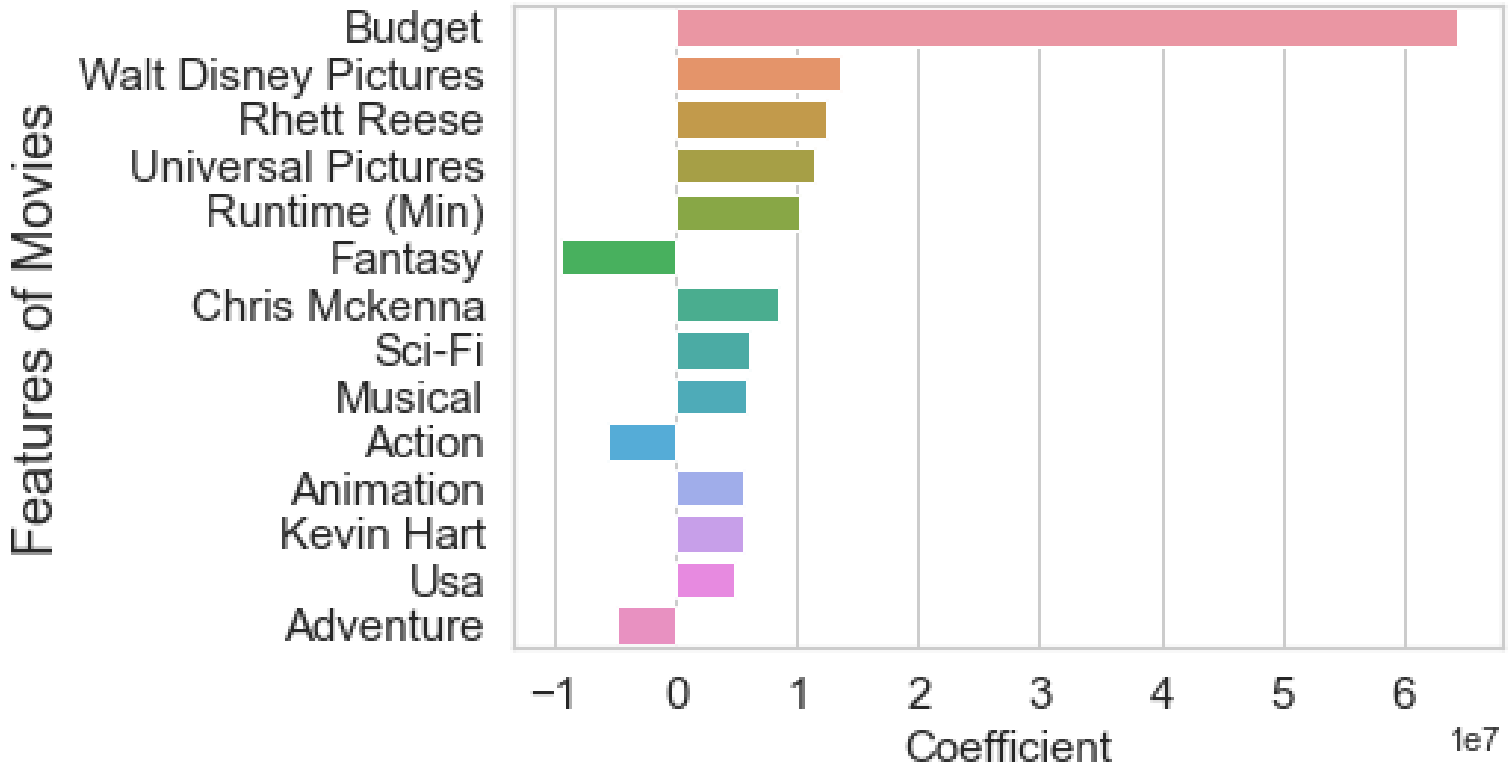




Left Plot: The points should be close to the line.
Middle Plot: The points should look random
Right Plot: The points should be close to the red line.

TOP 15 IMPORTANT FEATURES OF GROSS EARNING IN MOVIES

Budget and runtime are important.



IDEAL COMBINATION

Feature	
MPAA	R
Genre	Fantasy
Director	USA
Writer	Rhett Reese
Stars	Chris McKenna
Production company	Walt Disney Pictures
Country	USA
Release month	Jan

BEFORE COVID AND AFTER

- Lower gross USA after the start of covid

```
BeforeCovid=movies_df_drop.loc[movies_df_drop['years since release'] > 1.5]
BeforeCovid['gross usa'].mean()
```

```
64129600.560517035
```

Sixty million

```
AfterCovid=movies_df_drop.loc[movies_df_drop['years since release'] <= 1.5]
AfterCovid['gross usa'].mean()
```

```
468135.4
```

Four hundred thousand