Data Analysis Project 2
Introduction to Data Science, Section 001

Fall Semester, 2023
Group: DS 34
Members: Tanvi Bansal, Natasha Recoder, Tyler Perez

Q1 -

We imputed the dataset by filling in the nans of the movie ratings columns with a 50/50 blend of the mean of each column and row, as per the instructions. We used this data to run simple linear regression models where we used each movie's ratings to predict each individual other movie's ratings and calculated their CODs. For each movie we found the movie that predicted it the best and plotted those CODs in a histogram (Figure 1). Taking the average of these CODs resulted in an average best prediction COD of 0.4238 (Code 1).

We then calculated which 10 movies had the highest and lowest average CODs when that movie's ratings were being predicted by another movie, and which 10 movies had the highest average CODs when that movie's ratings were used as predictors (Table 1).

We did not cross validate thus we do not know if our model is over or under fit. If we had separated out some of the users we could then see if our models were under or overfit by comparing the CODs with the test and train sets.

Q2 -

Of the 10 movies that are best and least well predicted from the ratings of a single other movie we build multiple linear regressions models including gender identity, sibship status, social viewing preferences and the best predicting movie from part one. We found the max COD from all of linear regressions that predicted each movie to find the one movie that predicted each movie the best. We then compared those CODs and took the top 10 and the bottom 10. We did not use the movies from the average COD list, as it would not make sense to later compare an average COD from many linear regressions to one single R^2 from one multiple linear regression, however both interpretations of the problem statement were analyzed in code 1 (Appendix).

Given we did not have gender identity, social viewing preferences, and sibship status for every participant we removed the rows containing no gender identity information or no response for sibship status and social viewing preferences, given we could not interpolate a value and we cannot run the multiple linear regressions without this information. We understand removing these rows can add a confounding variable when comparing the multiple linear regression to the simple linear regression, however given the number of rows with nans or no response (33) was relatively small compared to the dataset size (1097) we decided to go forward with this method. R^2 went up in all cases, as expected given we were transitioning from single to multiple linear regression, and it did so in a linear fashion (Figure 2). Adding parameters will always increase the R^2, thus we should be aware that this increased R^2 doesn't necessarily translate to a superior model.

By adding so many parameters we decrease the amount of data in any one "box" which leads to the "coverage problem" and potential overfitting. We believe an issue with running this multiple regression model is the potential confounds between the variables. Sibship status could influence social viewing preferences for example. Overall we found running the simple linear regression model yielded better predictions than our multiple regression model (Code 1).

Q3 -

We began by picking the middle 30 movies from our sorted average CODs list. These movies ranged from a COD of .205 (Full Metal Jacket) to .185 (The Thing). We then selected 10 random other movies not from the

aforementioned list of 30, to build a Regularized Ridge Regression Model to predict the rating for the 30 movies. An important note is that the 10 predictor movies were randomly selected but remained consistent for each model.

We engaged in hyperparameter tuning to find the optimal lambda penalty term. We iterated through five different lambda values [0.01, 0.1, 1, 10, 100] on the 30 groups of movies paired with their 10 predictors. We then picked the model which yielded the lowest RMSE. At least one model utilized each of our 5 possible lambda values. The RMSE of our 30 models ranged from .276 for the movie The Man on Fire (Lambda=.01) to .507 for The Truman Show (Lambda=100.0) (Table 3). Our 10 predictor movies are listed in the appendix. All the betas were relatively small especially compared to our other models throughout this paper. All of the Betas were <1.0 and many of which were below .1. This aligns with our understanding of regularized regression models as they purposely shrink the beta coefficients.

The existence of confounding variables is a possible issue not being accounted for in our approach. Much of which can influence what one looks for when they watch a movie to attain enjoyment. Additionally, there are possible concerns regarding multicollinearity with using as many movie predictors as we do. An additional issue related to using so many predictors, and thus adding so many dimensions, is the coverage problem that arises, as mentioned in question 2.

Q4 -

Our Lasso Regression followed very much the same outline as our Ridge Regression. We selected the same 30 movies, and then selected the same 10 random movies for each movie. However we decided to take a more fine tuned approach in picking our lambda penalty hyperparameters. We used five fold cross validation and LassoCV to optimally tune our parameter to find the one which yielded the lowest RMSE. Oddly enough this does not yield much of a difference in RMSE compared to our previous ridge regression which chose one lambda from five possible ones.

The RMSE ranged from the lowest at .284 for Man on Fire once again to .513 for The Truman Show. The RMSEs were higher for both the min and max when compared to our ridge regression models, while the two movies remained the same (Table 3, 4). When looking at the Betas for each model one can see that many of the coefficients were 0.0, implying that these movies provide us no information about the movies in which we are trying to predict. For example Rocky (1978) had 3 different predictors whose beta coefficients were pushed down to 0. This provides us with simpler models and allows us to truly determine what movies are simply noise in predicting average ratings of the 30 selected movies. Through exploring our models we saw that the coefficients that were not dropped tended to be higher than that of ridge. Our Lambda penalty terms were all quite small all hovering around .01 or .1, this makes sense as when I followed the same approach as the previous question .1 yielded the lowest RMSE for each movie.

In conclusion though the RMSE did not change greatly from one form of regularization to another, yet the simplicity of the models changed drastically. Our Lambda parameters being so miniscule prompts concerns regarding overfitting (Table 4).

Q5 -

First we computed each user's average rating across all movies and each movie's average rating across all users utilizing element-wise removal of missing data to preserve maximal useful data. Next we selected the target movies and labeled the imputed user ratings for each target movie with 1 or 0 for enjoyed/not enjoyed. Then we fit the logistic regression model on the training set (split 50/50) utilizing the l2 penalty and inverse regularization strength c = 1.0

We chose a low c-value to accommodate overfitting risk present as our training data set is small (~500 points) which may not be representative of real world data. We chose the l2 penalty because our objective does not require sparsity or outlier management; we have only one feature of interest as our predictor and the input space is bounded so we are not concerned about outliers artificially inflating the penalty.

Table 5.1 and Figures 5.1 - 5.4 (Appendix) show the results of the model's predictive abilities on the test set. The resulting AUC values are strong implying no discrimination and favorable model accuracy. Betas are

relatively strong which implies stronger probability inflection and greater distance from decision boundary between classes, therefore lower chance of error as the sample size grows. We are still weary of overfitting however due to the small sample size - in the future we may consider increasing the survey size, simulating more data points, increasing the training proportion size of the sample, or increasing the regularization strength.

EC - Which movie franchise can best predict the ratings of another franchise?

For the extra credit we explored which movie franchise could best predict the enjoyment of another franchise. First we computed each user's average rating across each movie in a given franchise utilizing element-wise removal of missing data. Then we fit a ridge regression model between the training set (split 60/40) of the predicted and predictor franchise ratings utilizing the l2 penalty and regularization strength alpha = 100.0.

We chose the l2 penalty because our objective does not require sparsity as we have only one predictor feature for each model, and the input space is bounded so we don't have concerning outliers. We chose a middling alpha value as our training data set is small (~500 sample points) which presents overfitting risk as the training sample may not be representative of real world data, however our data's dimensions are already reduced.

Table EC and Figures EC.1 - EC.2 (Appendix) show the results of the model's predictive abilities on the test set. According to our model the enjoyment of the Star Wars franchise can best be predicted by the Indiana Jones franchise, however the resulting COD is weak (COD = 0.296) implying that much of the variation is not captured by the model. In the future we may consider increasing the sample size, increasing the training proportion of the sample size or reducing the regularization strength.

Figures

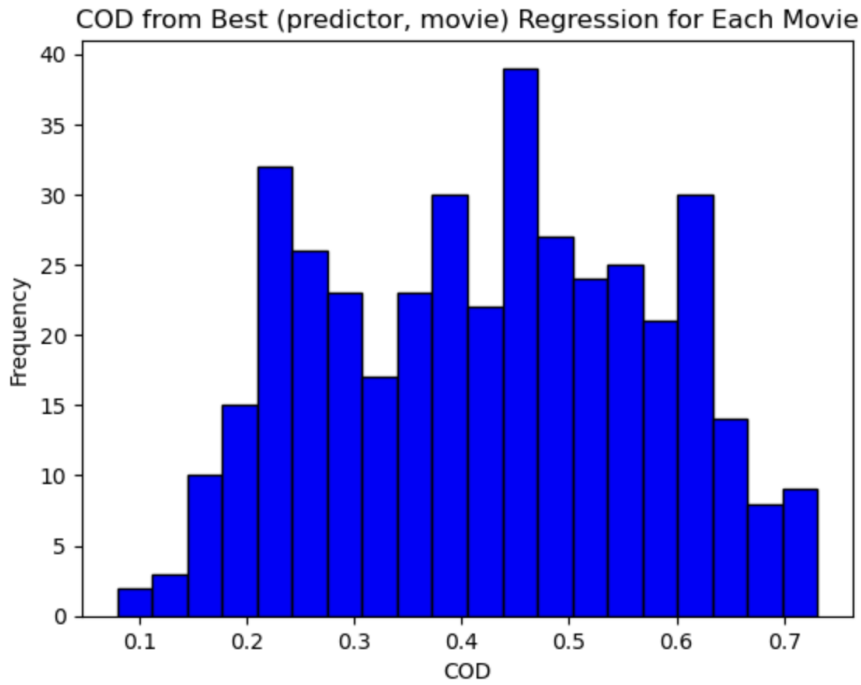


Figure 1- Histogram of the CODs of the movie that predicted each movie the best

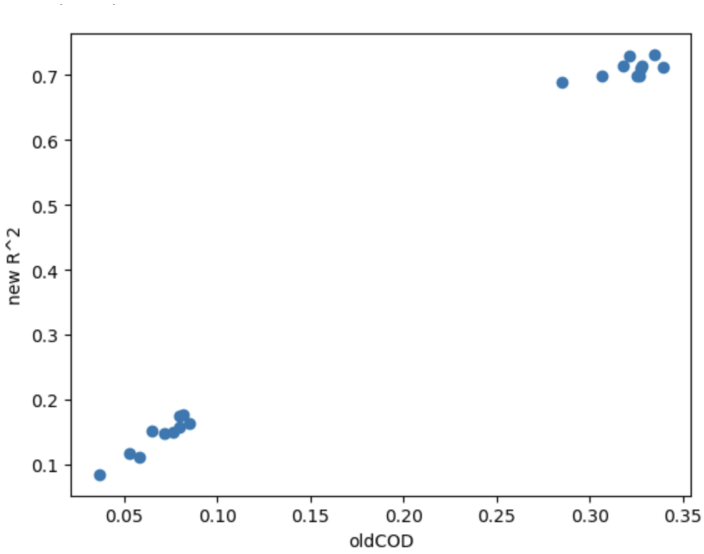


Figure 2- Scatter plot for the 10 movies that are best and least well predicted from the ratings of a single other movie, with the old COD from the simple linear regression on the x axis and the new R^2 from the multiple linea regression on the y axis

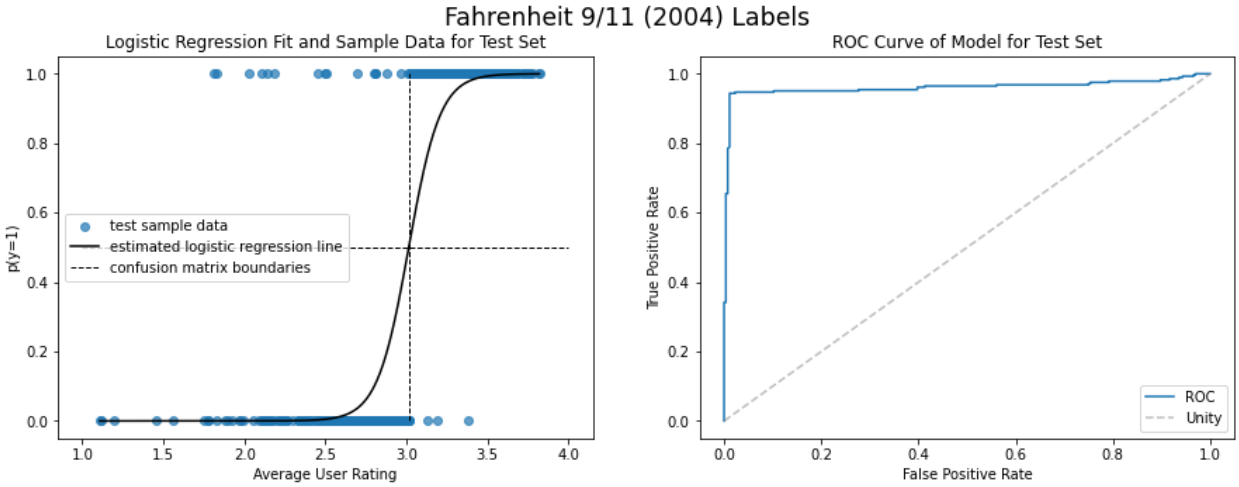


Figure 5.1 - Logistic regression fit for Farenheit 9/11 (left) and resulting ROC curve (right)

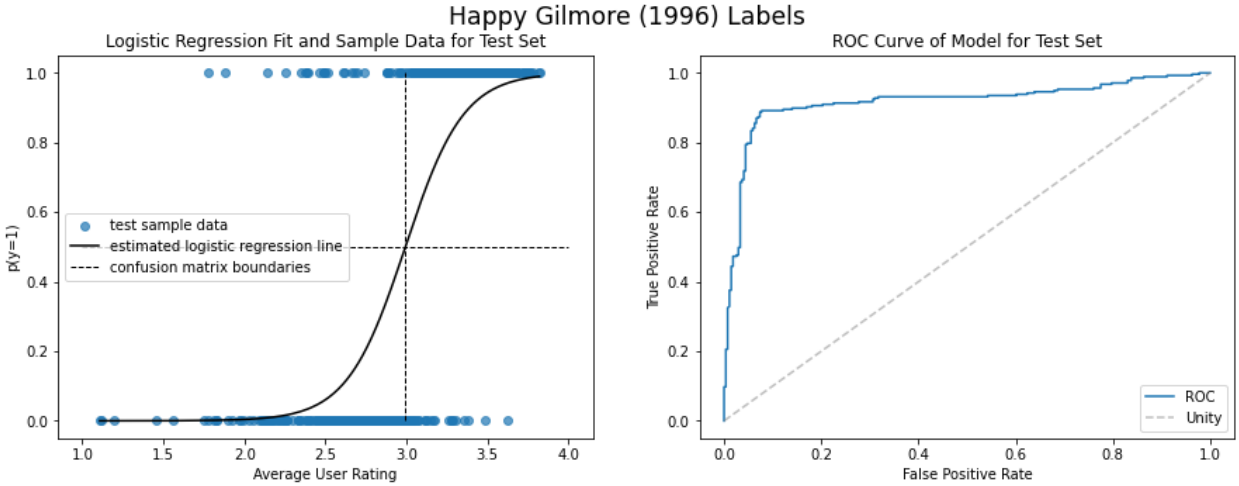


Figure 5.2 - Logistic regression fit for Happy Gilmore (left) and resulting ROC curve (right)

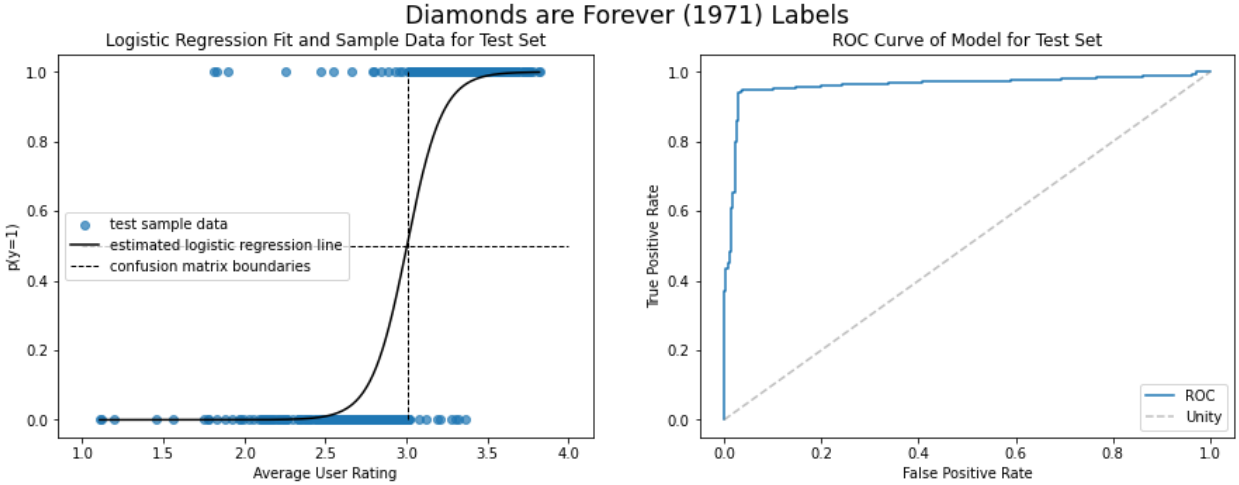


Figure 5.3 - Logistic regression fit for Diamonds are Forever (left) and resulting ROC curve (right)

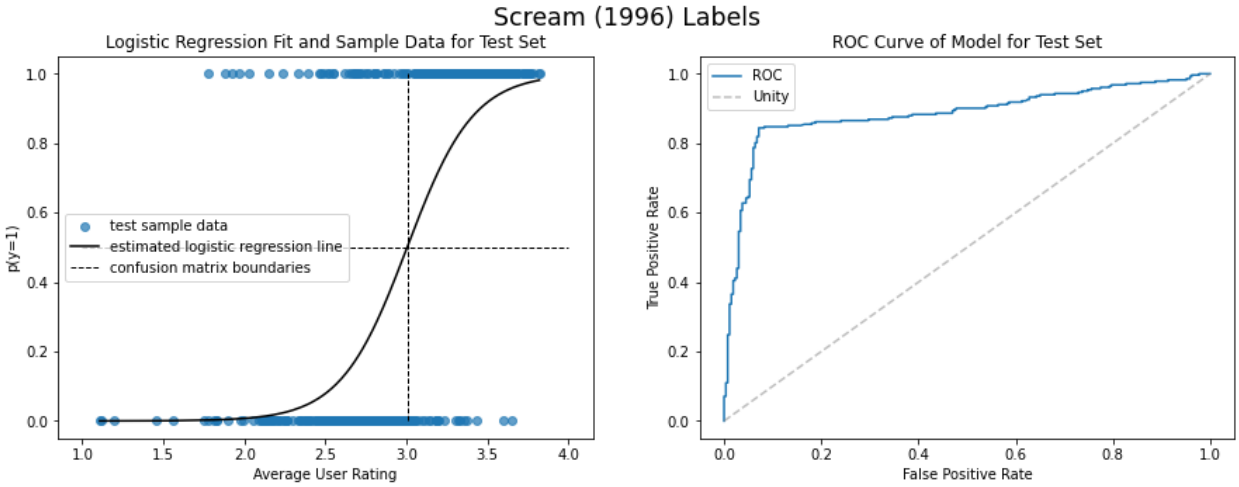


Figure 5.4 - Logistic regression fit for Scream (left) and resulting ROC curve (right)

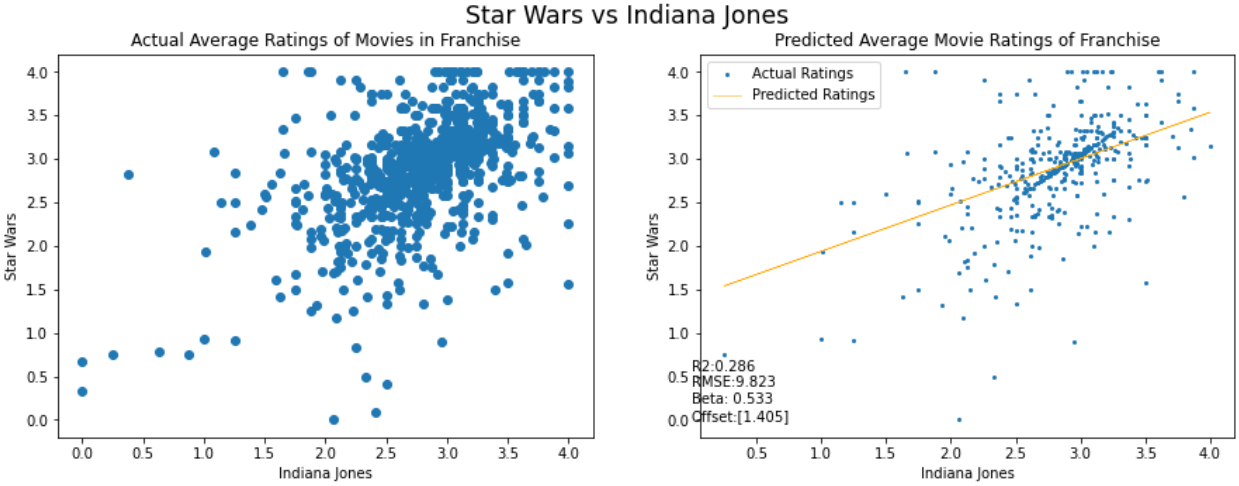


Figure EC.1 - Average ratings for Star Wars franchise vs. Indiana Jones franchise (left) and linear regression fit (right)

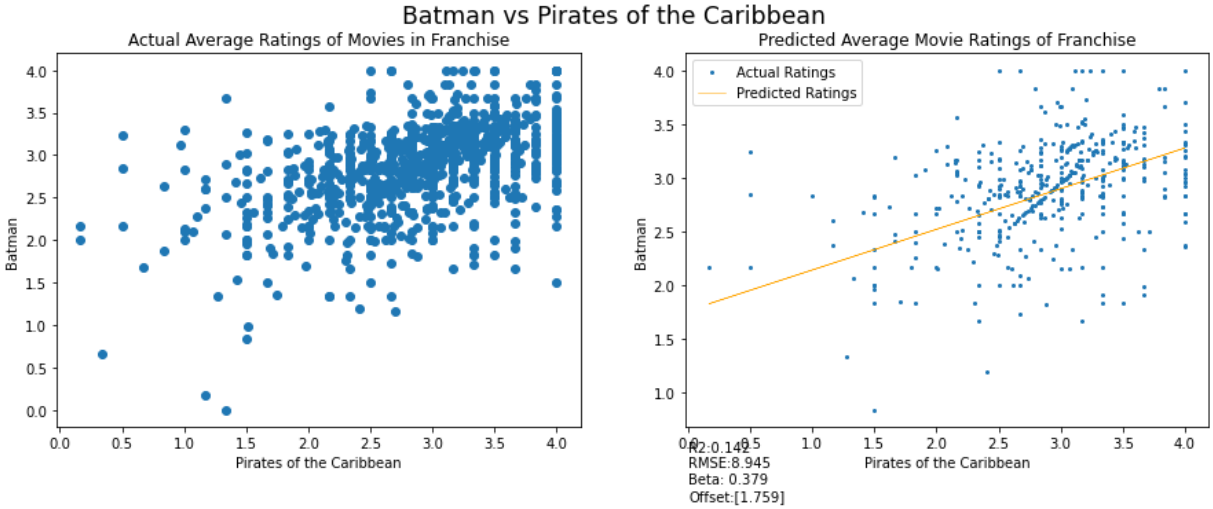


Figure EC.2 - Average ratings for Batman franchise vs. Pirates of the Caribbean franchise (left) and linear regression fit (right)

Predictor Movies for Q3 and Q4:
Goodfellas (1990)
Traffic (2000)
Boy's Don't Cry (1998)
Red Sonja (1985)
Just Like Heaven (2005)
Dead Poets Society (1989)
Battlefield Earth (2000)
The Perfect Storm (2000)
Mrs. Doubtfire (1993)
The Fast and Furious (2001)

Tables -

	Predicted	COD	Predictor
9	Escape from LA (1996)	0.343137	Sexy Beast (2000)
8	Sexy Beast (2000)	0.341748	The Silencers (1966)
7	The Lookout (2007)	0.339155	Patton (1970)
6	Erik the Viking (1989)	0.334651	I.Q. (1994)
5	Crimson Tide (1995)	0.330077	The Straight Story (1999)
4	The Bandit (1996)	0.327867	Best Laid Plans (1999)
3	Patton (1970)	0.327316	The Lookout (2007)
2	The Straight Story (1999)	0.326774	Congo (1995)
1	Miller's Crossing (1990)	0.326459	The Lookout (2007)
0	Heavy Traffic (1973)	0.325615	Ran (1985)
19	Shrek (2001)	0.075705	Shrek 2 (2004)
18	Pirates of the Caribbean: Dead Man's Chest (2006)	0.071986	Pirates of the Caribbean: At World's End (2007)
17	Clueless (1995)	0.071313	Escape from LA (1996)
16	The Avengers (2012)	0.069728	Captain America: Civil War (2016)
15	Shrek 2 (2004)	0.065492	Shrek (2001)
14	The Cabin in the Woods (2012)	0.065011	The Evil Dead (1981)
13	Black Swan (2010)	0.058106	Sorority Boys (2002)
12	Interstellar (2014)	0.052938	Torque (2004)
11	The Conjuring (2013)	0.042861	The Exorcist (1973)
10	Avatar (2009)	0.036669	Bad Boys (1995)

Table 1 - Movies that were on average able to be predicted the best, the average COD when linear regression was run with that movie being predicted and the predictor movie that predicted that movie best.

Movie	RMSE
Full Metal Jacket (1987)	0.388
Angles in the Outfield (1994)	0.398
You’re Next (2011)	0.363
The Green Mile (1999)	0.334
The Transporter (2002)	0.398
Child’s Play (1988)	0.395
Blues Brothers 2000 (1998)	0.338
Memento (2000)	0.401
Gone in Sixty Seconds (2000)	0.293
The Truman Show (1998)	0.507
Man on Fire (2004)	0.276
Crossroads (2002)	0.363
Knight and Day (2010)	0.377
Aliens (1986)	0.322
The Intouchables (2011)	0.354
Rocky (1976)	0.458
The Poseidon Adventure (1972)	0.454
Halloween (1978)	0.403
Big Daddy (1999)	0.298
Honey (2003)	0.389
One Flew Over the Cuckoo’s Nest (1975)	0.345
Baby Geniuses (1999)	0.378
Braveheart (1995)	0.346
The Others (2001)	0.401
The Mist (2007)	0.382
Bad Boys 2 (2003)	0.408

Armageddon (1998)	0.387
Bad Boys (1995)	0.340
12 Monkeys (1995)	0.377
The Thing (1982)	0.351

Table 3 - Ridge Regression Models RMSE

Movie	RMSE
Full Metal Jacket (1987)	0.387
Angles in the Outfield (1994)	0.401
You’re Next (2011)	0.364
The Green Mile (1999)	0.342
The Transporter (2002)	0.398
Child’s Play (1988)	0.401
Blues Brothers 2000 (1998)	0.348
Memento (2000)	0.403
Gone in Sixty Seconds (2000)	0.300
The Truman Show (1998)	0.513
Man on Fire (2004)	0.284
Crossroads (2002)	0.370
Knight and Day (2010)	0.380
Aliens (1986)	0.322
The Intouchables (2011)	0.355
Rocky (1976)	0.454
The Poseidon Adventure (1972)	0.464
Halloween (1978)	0.406
Big Daddy (1999)	0.302
Honey (2003)	0.390

One Flew Over the Cuckoo's Nest (1975)	0.343
Baby Geniuses (1999)	0.379
Braveheart (1995)	0.350
The Others (2001)	0.403
The Mist (2007)	0.383
Bad Boys 2 (2003)	0.426
Armageddon (1998)	0.392
Bad Boys (1995)	0.338
12 Monkeys (1995)	0.349
The Thing (1982)	0.357

Table 4 - Lasso Regression Models with RMSE

Movie	**Fahrenheit 9/11**	**Happy Gilmore**	**Diamonds are Forever**	**Scream**
Beta0	-31.0617	-16.6288	-27.3469	-14.6318
Beta1	10.3055	5.5560	9.1077	4.8707
AUC	0.9605	0.9166	0.9615	0.8849

Table 5.1 - Logistic regression fit metrics for all target movies in Q5

Predicted Franchise	**Predictor Franchise**	**Superlative**	**Value**
Star Wars	Indiana Jones	Highest COD	0.297
Star Wars	Indiana Jones	Lowest RMSE	9.649
Batman	Pirates of Caribbean	Highest Beta	0.681

Table EC - Linear regression fit metrics of interest for franchises studied in EC

CODE 1

In [1]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [2]:
```
df = pd.read_csv('../project1/movieReplicationSet.csv')
```

Imputation

In [3]:
```
df = df.iloc[:,0:400]
```

In [4]:
```
col_means = df.mean()
row_means = df.mean(axis=1)
```

In [5]:
```
df_filled = df.apply(lambda col: col.fillna((col.mean() + row_means) / 2
```

In [6]:
```
df = df_filled.iloc[:,0:400]
```

In [7]:
```
df.shape
```

Out[7]: (1097, 400)

In [8]:
```
df
```

Out[8]:

	The Life of David Gale (2003)	Wing Commander (1999)	Django Unchained (2012)	Alien (1979)	Indiana Jones and the Last Crusade (1989)	Snatch (2000)	Rambo: First Blood Part II (1985)	Fargo (1996)	Let t Rig One (20(
0	2.447086	2.381992	4.000000	2.725235	3.000000	2.670257	2.554121	2.821232	2.6196
1	2.439294	2.374200	1.500000	2.717443	2.752945	2.662464	2.546329	2.813440	2.6118
2	2.733065	2.667971	3.234118	3.011214	3.046716	2.956236	2.840100	3.107211	2.9055
3	2.282975	2.217880	2.000000	2.561123	3.000000	2.506145	2.390009	2.657120	2.4554
4	2.209132	2.144038	3.500000	2.487281	0.500000	2.432303	0.500000	1.000000	2.3816
...	...	...	...	...	...	...	...	...	
1092	2.675658	2.610563	3.176711	2.953806	3.500000	2.898828	2.782692	3.049803	2.8481

Checking that this worked with specific examples

In [9]:
```
(row_means[0]+col_means[0])/2 == df_filled.iloc[0,0]
```

```
/var/folders/q9/dzb_81254m11n5lj37dj2pk40000gn/T/ipykernel_61950/3919642532.py:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  (row_means[0]+col_means[0])/2 == df_filled.iloc[0,0]
```

Out[9]:
```
True
```

In [10]:
```
(row_means[2]+col_means[4])/2 == df_filled.iloc[2,4]
```

```
/var/folders/q9/dzb_81254m11n5lj37dj2pk40000gn/T/ipykernel_61950/3097046183.py:1: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  (row_means[2]+col_means[4])/2 == df_filled.iloc[2,4]
```

Out[10]:
```
True
```

In [11]:
```
#drop nan row

df.iloc[896,:]
```

Out[11]:
```
The Life of David Gale (2003)                NaN
Wing Commander (1999)                        NaN
Django Unchained (2012)                      NaN
Alien (1979)                                 NaN
Indiana Jones and the Last Crusade (1989)    NaN
                                              ..
Patton (1970)                                NaN
Anaconda (1997)                              NaN
Twister (1996)                               NaN
MacArthur (1977)                             NaN
Look Who's Talking (1989)                    NaN
Name: 896, Length: 400, dtype: float64
```

In [12]:
```
df = df.drop(896, axis = 0)
```

Linear Regression Part

In [13]:
```
#to store our vals
CODs = {}
predictorCODs = {}
```

In [14]:
```
#for each movie
for j in range(df.shape[1]):
    #this is the name of the movie we are predicting for
    movie_topred_name = df.columns[j]
    movie_topred = df.iloc[:, j].values.reshape(-1,1)
    CODs[movie_topred_name] = {}
    #we remove that movie from the dataframe
    sub_df = df.drop(movie_topred_name, axis =1)
    for i in range(sub_df.shape[1]):
        #now we choose one of teh others as a predictor
        predictor_name = sub_df.columns[i]
        predictorCODs[predictor_name] = {}
        predictor = sub_df.iloc[:, i].values.reshape(-1,1)
        reg = LinearRegression().fit(predictor, movie_topred)
        y_hat = reg.predict(predictor)
        COD = r2_score(movie_topred,y_hat)
        #store the COD
        CODs[movie_topred_name][predictor_name] = COD
```

CODs dictionary stored in manner of predicted movie: {predictor, COD}

In [15]:
```
bestCODs = {}
averageCODs = {}
```

In [16]:
```
#To get best cods as a nested dict
for key in CODs:
    #to get the movie that predicts it the best
    bestCOD = {}
    bestCOD[max(CODs[key].items(), key=lambda x: x[1])[0]] = max(CODs[ke
    bestCODs[key] = bestCOD
    #to get average
    averageCODs[key]= (sum(CODs[key].values())/len(CODs[key]))
```

In [17]:
```
bestCODs['The Final Conflict (1981)']
```

Out[17]:
```
{'The Lookout (2007)': 0.7001881161214467}
```

In [18]:
```
bestCODs['The Lookout (2007)']
```

Out[18]:
```
{'Patton (1970)': 0.7135542589926913}
```

In [19]:
```
predictor = sub_df['The Final Conflict (1981)'].values.reshape(-1,1)
movie_topred = df['The Lookout (2007)'].values.reshape(-1,1)
reg = LinearRegression().fit(predictor, movie_topred)
y_hat = reg.predict(predictor)
COD = r2_score(movie_topred,y_hat)
```

In [20]:
```
COD
```

Out[20]:
```
0.7001881161214467
```

In [21]:
```
movie_topred  = sub_df['The Final Conflict (1981)'].values.reshape(-1,1)
predictor = df['The Lookout (2007)'].values.reshape(-1,1)
reg = LinearRegression().fit(predictor, movie_topred)
y_hat = reg.predict(predictor)
COD = r2_score(movie_topred,y_hat)
```

In [22]:
```
COD
```

Out[22]:
```
0.7001881161214467
```

In [23]:
```
#DataFrame of best
data = [(key1, key2, value) for key1, inner_dict in bestCODs.items() for

# Creating a DataFrame
predictedBest = pd.DataFrame(data, columns=['Predicted', 'Predictor', 'C(
```

getting the average of the best CODS

In [24]:
```
predictedBest.sort_values('COD', ascending = False).head(10)
```

Out[24]:

	Predicted	Predictor	COD
203	Erik the Viking (1989)	I.Q. (1994)	0.731507
208	I.Q. (1994)	Erik the Viking (1989)	0.731507
395	Patton (1970)	The Lookout (2007)	0.713554
377	The Lookout (2007)	Patton (1970)	0.713554
240	The Bandit (1996)	Best Laid Plans (1999)	0.711222
249	Best Laid Plans (1999)	The Bandit (1996)	0.711222
282	Congo (1995)	The Straight Story (1999)	0.700569
287	The Straight Story (1999)	Congo (1995)	0.700569
334	The Final Conflict (1981)	The Lookout (2007)	0.700188
300	Ran (1985)	Heavy Traffic (1973)	0.692734

In [25]:
```
#each movie and its predicted
predictedBest['COD'].mean()
```

Out[25]:
```
0.42378171067196035
```

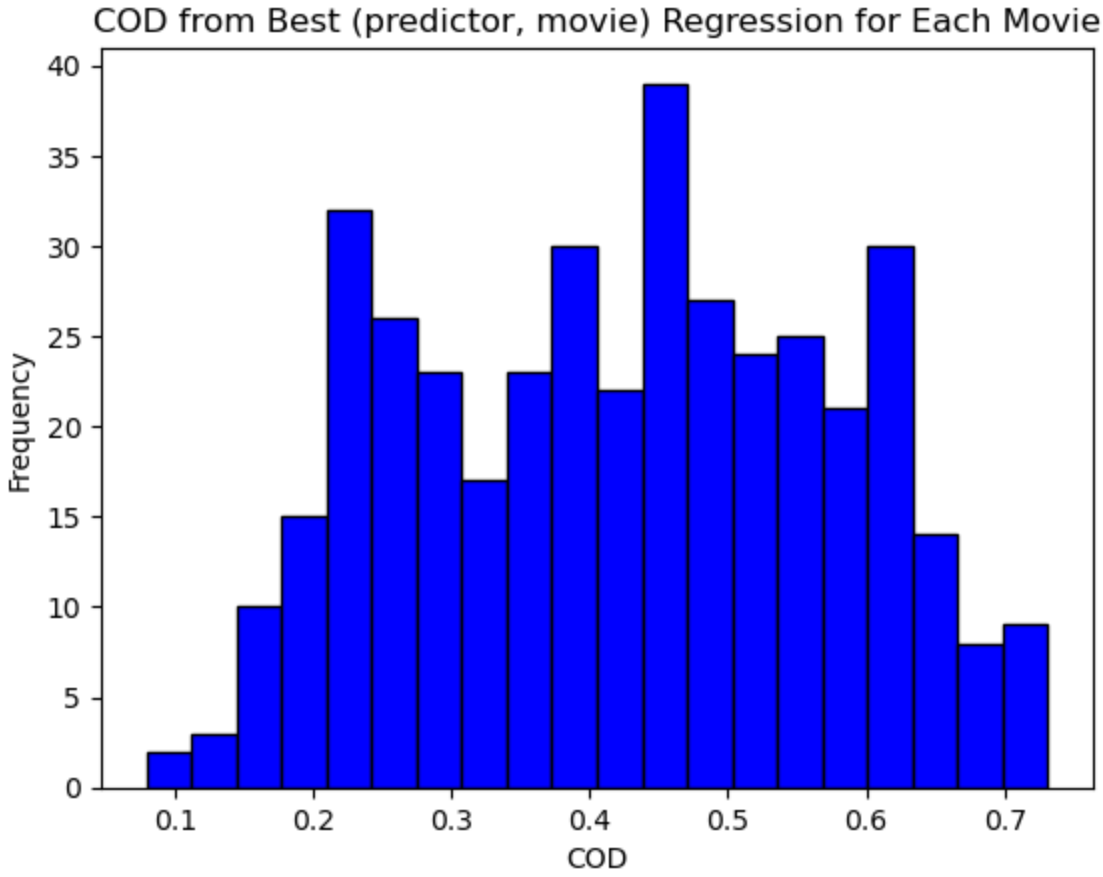
In [26]:
```
#histogram of our 400 cod values
values = list(predictedBest['COD'])

# Plot histogram
plt.hist(values, bins=20, color='blue', edgecolor='black')

# Add labels and title
plt.xlabel('COD')
plt.ylabel('Frequency')

plt.title('COD from Best (predictor, movie) Regression for Each Movie')

# Show the plot
plt.show()
```



Goal of getting the average COD of each movie

In [27]:
```
#DataFrame of ALL CODS
data = [(key1, key2, value) for key1, inner_dict in CODs.items() for key2
# Creating a DataFrame
allCODs = pd.DataFrame(data, columns=['Predicted', 'Predictor', 'COD'])
```

In [28]:
```
#if we take the average ability of each movie at being a predictor
predictorAverages = {}
for movie_name in CODs.keys():
    predictorAverages[movie_name] = allCODs[allCODs['Predictor'] == movi
```

In [29]:
```
predictors = pd.DataFrame(list(predictorAverages.items()), columns=['Pre
```

In [30]:
```
#10 movies best at being predictors on average
bestPredictors = predictors.sort_values('Average COD', ascending = False
```

In [31]:
```
bestPredictors
```

Out[31]:

	Predictor	Average COD
116	Escape from LA (1996)	0.343137
109	Sexy Beast (2000)	0.341748
377	The Lookout (2007)	0.339155
203	Erik the Viking (1989)	0.334651
298	Crimson Tide (1995)	0.330077
240	The Bandit (1996)	0.327867
395	Patton (1970)	0.327316
287	The Straight Story (1999)	0.326774
363	Miller's Crossing (1990)	0.326459
309	Heavy Traffic (1973)	0.325615

In [32]:
```
bestPredictorsList = [f'{row["Predictor"]};{row["Average COD"]}' for ind
```

In [33]:
```
predictedAvgCODs = pd.DataFrame(averageCODs.items(), columns = ['Predict
```

In [34]:
```
#10 best at being predicted on average
bestPredicted = predictedAvgCODs.sort_values('Average COD', ascending = 
```

In [35]:
```
bestPredicted
```

Out[35]:

	Predicted movie	Average COD
116	Escape from LA (1996)	0.343137
109	Sexy Beast (2000)	0.341748
377	The Lookout (2007)	0.339155
203	Erik the Viking (1989)	0.334651
298	Crimson Tide (1995)	0.330077
240	The Bandit (1996)	0.327867
395	Patton (1970)	0.327316
287	The Straight Story (1999)	0.326774
363	Miller's Crossing (1990)	0.326459
309	Heavy Traffic (1973)	0.325615

In [36]:
```
bestPredictedList = [f'{row["Predicted movie"]};{row["Average COD"]}' fo
```

In [37]:
```
bestPredictedList
```

Out[37]:
```
['Escape from LA (1996);0.34313725950310314',
 'Sexy Beast (2000);0.341748374518728',
 'The Lookout (2007);0.3391550604987197',
 'Erik the Viking (1989);0.3346512445380443',
 'Crimson Tide (1995);0.3300768990189199',
 'The Bandit (1996);0.32786717386930514',
 'Patton (1970);0.3273164658859912',
 'The Straight Story (1999);0.32677400903723974',
 "Miller's Crossing (1990);0.32645924019042405",
 'Heavy Traffic (1973);0.3256150943231005']
```

In [38]:
```
#worst predicted
worstPredicted = predictedAvgCODs.sort_values('Average COD').head(10)
```

In [39]:
```
worstPredictedList = [f'{row["Predicted movie"]};{row["Average COD"]}' f
```

In [40]:
```
#Change TABLE
```

Table

In [41]:
```
_df = pd.DataFrame(CODs)
_avg = np.mean(CODS_df,axis=0)
_max = np.max(CODS_df,axis=0)

_max_predictor = []
i in range(len(CODS_max)):
predicted_title = CODS_max.index[i]
predicted_COD_val = CODS_max[i]
predictor_title = CODS_df[predicted_title][CODS_df[predicted_title] == pr
CODS_max_predictor.append({"Predicted":predicted_title, "MaxCOD": predict

_max_predictor = pd.DataFrame(CODS_max_predictor)
_max_predictor = CODS_max_predictor.sort_values(by=['MaxCOD'])

_avg_predictor = CODS_avg.sort_values()

om_predicted = CODS_avg_predictor[0:10]
predicted = CODS_avg_predictor[-10:]

_worst_predicted = []
i in range(10):
predicted = top_predicted.index[i]
cod = top_predicted[i]
predictor_ind = np.argwhere([CODS_max_predictor['Predicted'] == top_predi
predictor = CODS_max_predictor.iloc[predictor_ind]['MaxPredictor']
best_worst_predicted.append({"Predicted":predicted, "COD":cod, "Predictor
i in range(10):
predicted = bottom_predicted.index[i]
cod = bottom_predicted[i]
predictor_ind = np.argwhere([CODS_max_predictor['Predicted'] == bottom_pr
predictor = CODS_max_predictor.iloc[predictor_ind]['MaxPredictor']
best_worst_predicted.append({"Predicted":predicted, "COD":cod, "Predictor

_worst_predicted = pd.DataFrame(best_worst_predicted)
_worst_predicted = best_worst_predicted.sort_values(by=['COD'],ascending=
```

```
/var/folders/q9/dzb_81254m11n5lj37dj2pk40000gn/T/ipykernel_61950/2528961847.py:8: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  predicted_COD_val = CODS_max[i]
/var/folders/q9/dzb_81254m11n5lj37dj2pk40000gn/T/ipykernel_61950/2528961847.py:23: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  cod = top_predicted[i]
/var/folders/q9/dzb_81254m11n5lj37dj2pk40000gn/T/ipykernel_61950/2528961847.py:29: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  cod = bottom_predicted[i]
```

In [42]:
```
best_worst_predicted
```

Out[42]:

	Predicted	COD	Predictor
9	Escape from LA (1996)	0.343137	Sexy Beast (2000)
8	Sexy Beast (2000)	0.341748	The Silencers (1966)
7	The Lookout (2007)	0.339155	Patton (1970)
6	Erik the Viking (1989)	0.334651	I.Q. (1994)
5	Crimson Tide (1995)	0.330077	The Straight Story (1999)
4	The Bandit (1996)	0.327867	Best Laid Plans (1999)
3	Patton (1970)	0.327316	The Lookout (2007)
2	The Straight Story (1999)	0.326774	Congo (1995)
1	Miller's Crossing (1990)	0.326459	The Lookout (2007)
0	Heavy Traffic (1973)	0.325615	Ran (1985)
19	Shrek (2001)	0.075705	Shrek 2 (2004)
18	Pirates of the Caribbean: Dead Man's Chest (2006)	0.071986	Pirates of the Caribbean: At World's End (2007)
17	Clueless (1995)	0.071313	Escape from LA (1996)
16	The Avengers (2012)	0.069728	Captain America: Civil War (2016)
15	Shrek 2 (2004)	0.065492	Shrek (2001)
14	The Cabin in the Woods (2012)	0.065011	The Evil Dead (1981)
13	Black Swan (2010)	0.058106	Sorority Boys (2002)
12	Interstellar (2014)	0.052938	Torque (2004)
11	The Conjuring (2013)	0.042861	The Exorcist (1973)
10	Avatar (2009)	0.036669	Bad Boys (1995)

QUESTION 2

In [43]:
```
og_df = pd.read_csv('../project1/movieReplicationSet.csv')
```

In [44]:
```
og_df.keys()
```

Out[44]:
```
Index(['The Life of David Gale (2003)', 'Wing Commander (1999)',
       'Django Unchained (2012)', 'Alien (1979)',
       'Indiana Jones and the Last Crusade (1989)', 'Snatch (2000)',
       'Rambo: First Blood Part II (1985)', 'Fargo (1996)',
       'Let the Right One In (2008)', 'Black Swan (2010)',
       ...
       'When watching a movie I cheer or shout or talk or curse at the
screen',
       'When watching a movie I feel like the things on the screen are
happening to me',
       'As a movie unfolds I start to have problems keeping track of ev
ents that happened earlier',
       'The emotions on the screen "rub off" on me - for instance if so
mething sad is happening I get sad or if something frightening is happe
ning I get scared',
       'When watching a movie I get completely immersed in the alternat
ive reality of the film',
       'Movies change my position on social economic or political issue
s',
       'When watching movies things get so intense that I have to stop
watching',
       'Gender identity (1 = female; 2 = male; 3 = self-described)',
       'Are you an only child? (1: Yes; 0: No; -1: Did not respond)',
       'Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respo
nd)'],
      dtype='object', length=477)
```

In [45]:
```
#same for all
gender = og_df['Gender identity (1 = female; 2 = male; 3 = self-describe
siblings = og_df['Are you an only child? (1: Yes; 0: No; -1: Did not res
social = og_df['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did no
```

In [46]:
```
gender = np.array(gender).reshape(-1,1)
```

In [47]:
```
to_rem = []
for i in range(len(gender)):
    if (np.isnan(gender[i]) == True):
        to_rem.append(i)
for i in range(len(siblings)):
    if ((siblings[i]) == -1):
        if i not in to_rem:
            to_rem.append(i)
for i in range(len(social)):
    if ((social[i]) == -1):
        if i not in to_rem:
            to_rem.append(i)
```

In [48]:
```
#the one of all nans
to_rem.remove(896)
```

In [49]:
```
#this has all the rows with incomplete data that we will remove
to_rem
```

Out[49]:
```
[27,
 269,
 273,
 295,
 316,
 371,
 454,
 460,
 686,
 831,
 844,
 858,
 882,
 902,
 905,
 923,
 947,
 971,
 972,
 973
```

remove rows with nan gender or no response other cols

In [50]:
```
len(to_rem)
```

Out[50]:
```
32
```

In [51]:
```
#remove the all nans column
og_df = og_df.drop(896, axis = 0)
```

In [52]:
```
#the original full dataframe remove the people with nan gender, etc
rem_df = og_df.drop(to_rem)
```

In [53]:
```
#the imputed dataframe remove the people with nan gender, etc
#weve already droped the nan columnm
rem_df_y = df.drop(to_rem)
```

In [54]:
```
gender = rem_df['Gender identity (1 = female; 2 = male; 3 = self-describ
sibship = rem_df['Are you an only child? (1: Yes; 0: No; -1: Did not resp
social = rem_df['Movies are best enjoyed alone (1: Yes; 0: No; -1: Did n
gender = np.array(gender).reshape(-1,1)
sibship = np.array(sibship).reshape(-1,1)
social = np.array(social).reshape(-1,1)
```

merge the lists and run the mult lin regression

In [55]:
```
#using 10 best and worst rom avg
toRegress = list(best_worst_predicted['Predicted'])
```

In [56]:
```
#using 10 from best
#THIS IS WHAT WE WENT WITH MAKES MORE SENSE WHEN PREDICTION COMES UP
toRegress = list(predictedBest.sort_values('COD',ascending = False).head
```

In [57]:
```
#new df with movie and new cod
multCOD = {}
for i in range(len(toRegress)):
    #movie ratings
    movName = toRegress[i]
    y = rem_df_y[movName]
    predictor = predictedBest[(predictedBest['Predicted'] == movName)]['
    predictorRatings = rem_df_y[predictor]
    x = np.concatenate((gender, sibship, social, predictorRatings), axis=
    reg = LinearRegression().fit(x,y)
    y_hat = reg.predict(x)
    COD = r2_score(y,y_hat)
    multCOD[movName] = COD
```

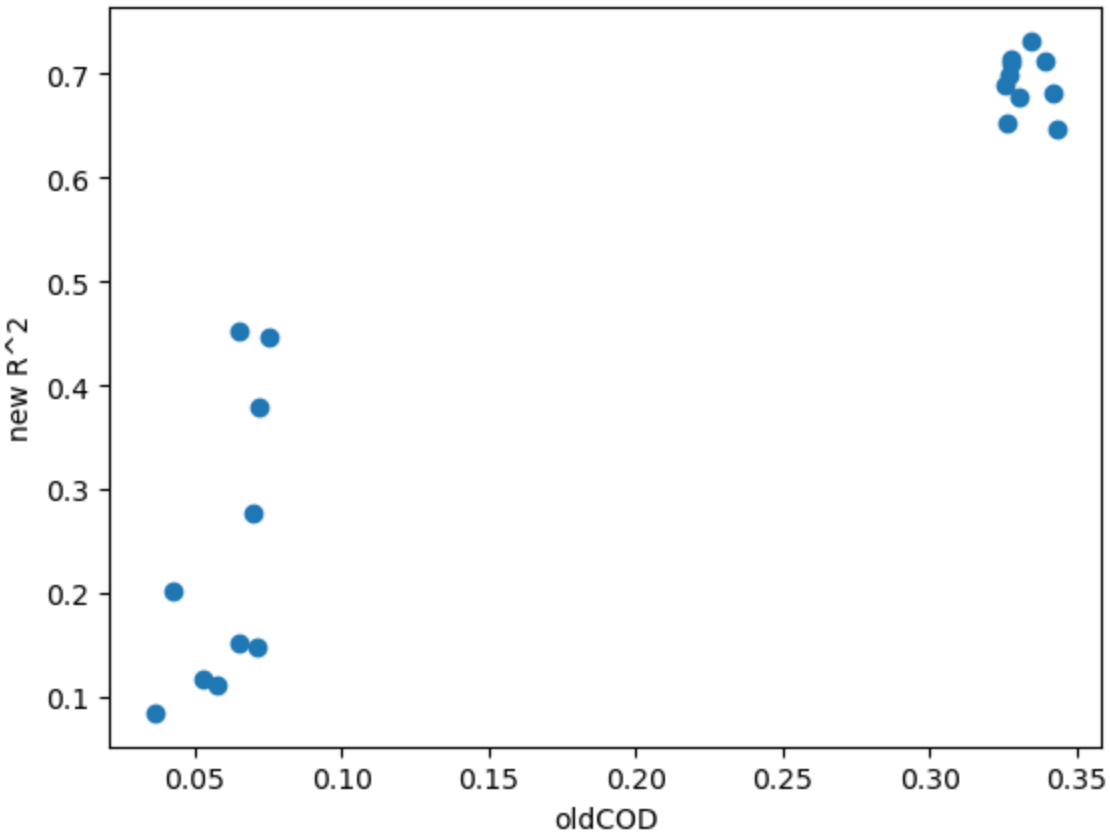
In [58]:
```
multCOD
```

Out[58]:
```
{'Erik the Viking (1989)': 0.7318748495913724,
 'I.Q. (1994)': 0.73060285808651,
 'Patton (1970)': 0.7112816133568812,
 'The Lookout (2007)': 0.7122370520219796,
 'The Bandit (1996)': 0.7152479020018117,
 'Best Laid Plans (1999)': 0.7150801283334667,
 'Congo (1995)': 0.6987123799427901,
 'The Straight Story (1999)': 0.6997635075519948,
 'The Final Conflict (1981)': 0.6984795773807653,
 'Ran (1985)': 0.6899047838258976,
 'Avatar (2009)': 0.0843163106360948,
 'Interstellar (2014)': 0.11673581103168296,
 'Black Swan (2010)': 0.11164845101737697,
 'Clueless (1995)': 0.14858962104322682,
 'The Cabin in the Woods (2012)': 0.1520318469225972,
 'La La Land (2016)': 0.14941274276043648,
 'Titanic (1997)': 0.15684858761152187,
 '13 Going on 30 (2004)': 0.16266644997722712,
 'The Fast and the Furious (2001)': 0.17483618535231027,
 'Grown Ups 2 (2013)': 0.1761770878203412 5}
```

In [59]:
```
# make
oldCODs = []
for movie in multCOD:
    oldCODs.append(averageCODs[movie])
```

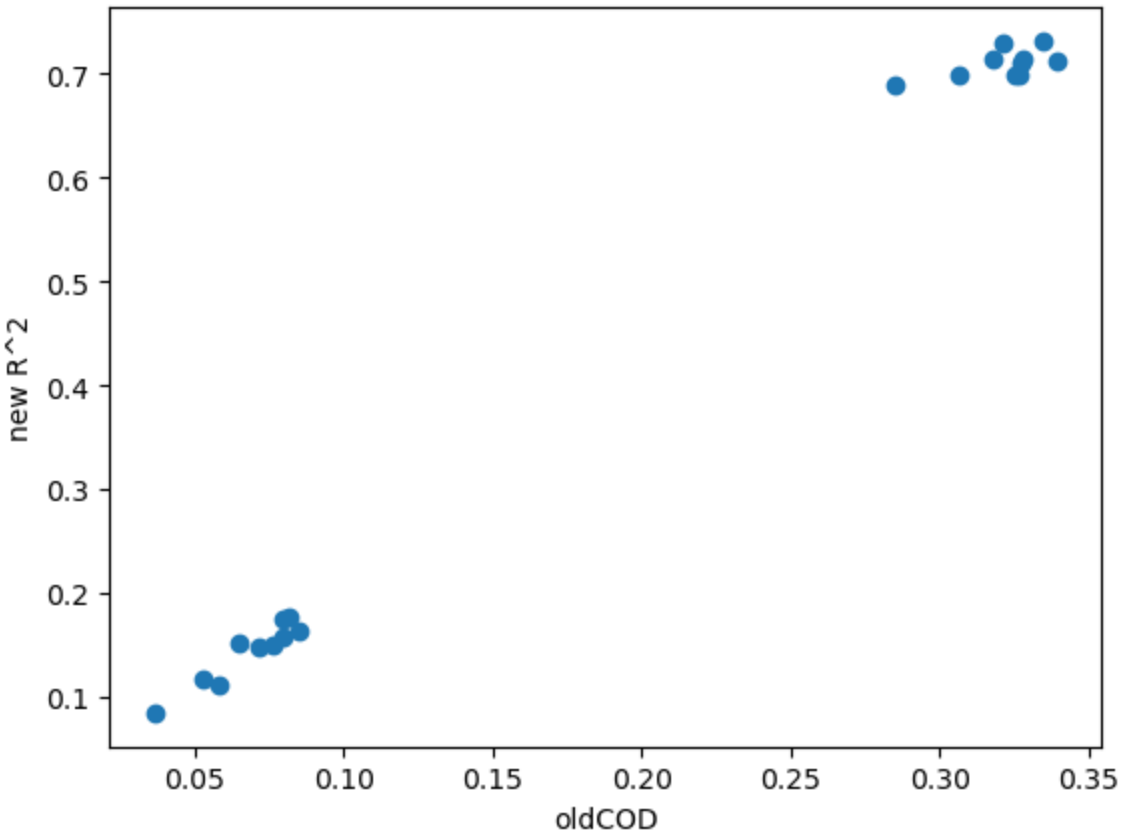
In [102]:
```
#from avg
plt.scatter(oldCODs, multCOD.values())
plt.xlabel('oldCOD')
plt.ylabel('new R^2')
```

Out[102]:
```
Text(0, 0.5, 'new R^2')
```



In [60]:
```
#from best
plt.scatter(oldCODs, multCOD.values())
plt.xlabel('oldCOD')
plt.ylabel('new R^2')
```

Out[60]: Text(0, 0.5, 'new R^2')



In [61]:
```
compareCODs = pd.DataFrame({'oldCod': oldCODs, 'multCOD': multCOD.values
```

In [62]:
```
compareCODs
```

Out[62]:

	oldCod	multCOD
0	0.334651	0.731875
1	0.321188	0.730603
2	0.327316	0.711282
3	0.339155	0.712237
4	0.327867	0.715248
5	0.318037	0.715080
6	0.306091	0.698712
7	0.326774	0.699764
8	0.325478	0.698480
9	0.285129	0.689905
10	0.036669	0.084316
11	0.052938	0.116736
12	0.058106	0.111648
13	0.071313	0.148590
14	0.065011	0.152032
15	0.076237	0.149413
16	0.079965	0.156849
17	0.084810	0.162666
18	0.079478	0.174836
19	0.081827	0.176177

In [ ]:
```

```