DS-GA 1001 Project 2 Report

Kunkun Hu, Jiaran Peng

Missing Data Handling Part:

We divided missing values into two categories when it comes to imputing missing values. The first 400 columns are movie rating columns. We chose to use half-column mean and half-row mean as instructed. That way can incorporate both row information and column information. One thing worth noting is that we found that there is a user who had not seen a single one of the 400 movies. We chose to impute missing values completely based on column mean. The other is to deal with the Gender identity column which is a categorical column. In that case, the method of using half-column mean and half-row mean does not make sense at all. We found that 80% of these column values are female. Therefore, we chose to impute with the most common value.

Question One:

In this question, we conducted 399 simple linear regressions for each of the 400 movies using the rating of other movies. The goal of simple linear regressions is to find the best-fitting straight line that represents the relationship between the variables. We recorded the best predictor and corresponding COD for each movie. Based on the histogram below, it looks roughly normally distributed with the mean of 0.4.

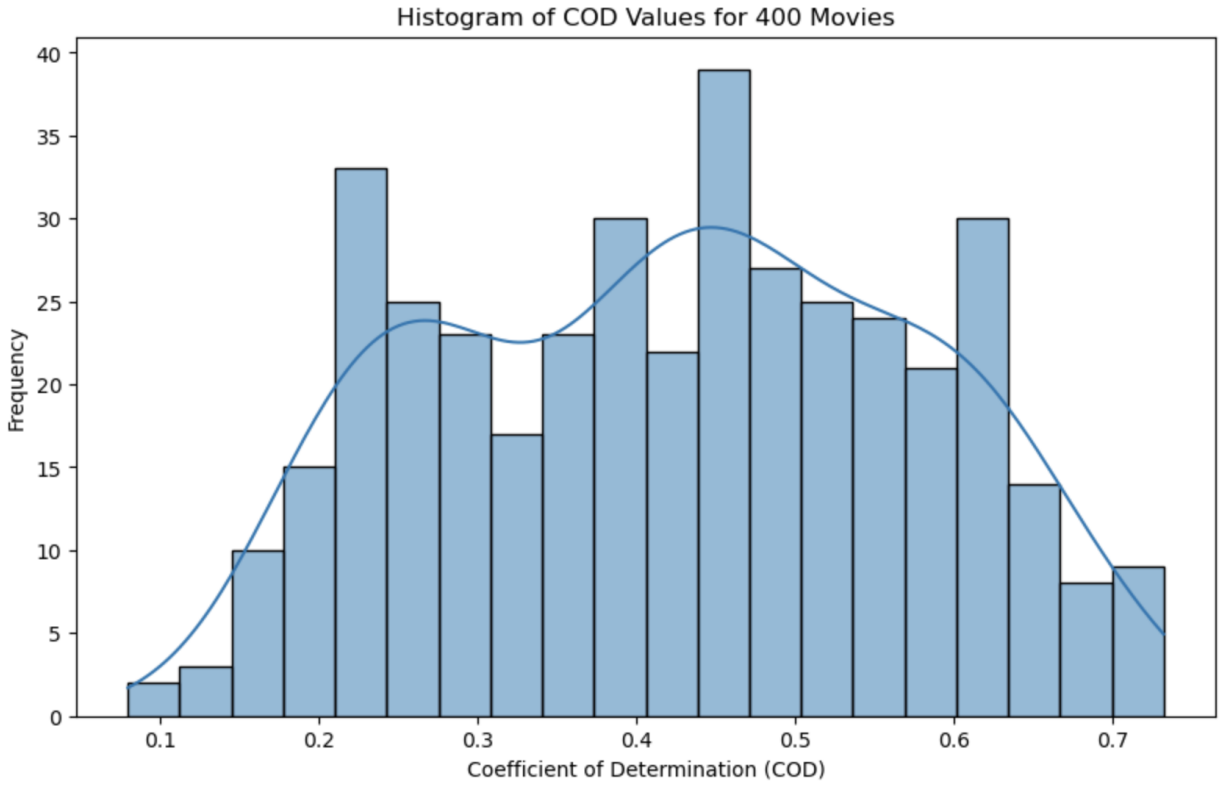


Figure One: histogram of COD for 400 Movies

We also constructed a dataframe that contains the top 10 easiest-to-predict movies and hardest-to-predict movies based on their best COD values. COD is a measurement of the proportion of variance in the dependent variable that is explained by the independent variables. A higher COD means it is easier to predict and vice versa. Therefore, the dataframe contains the movies with the 10 highest and the 10 lowest COD values. We also calculated the mean COD value of these 400 movies which is roughly 0.42. It means 42 percent of the variance is explained by the best-explaining movie on average.

	movie	best_COD	best_predictor
0	Erik the Viking (1989)	0.731789	I.Q. (1994)
1	I.Q. (1994)	0.731789	Erik the Viking (1989)
2	The Lookout (2007)	0.713793	Patton (1970)
3	Patton (1970)	0.713793	The Lookout (2007)
4	The Bandit (1996)	0.711540	Best Laid Plans (1999)
5	Best Laid Plans (1999)	0.711540	The Bandit (1996)
6	Congo (1995)	0.700822	The Straight Story (1999)
7	The Straight Story (1999)	0.700822	Congo (1995)
8	The Final Conflict (1981)	0.700437	The Lookout (2007)
9	Ran (1985)	0.692863	Heavy Traffic (1973)
19	Grown Ups 2 (2013)	0.171151	The Core (2003)
18	The Fast and the Furious (2001)	0.169000	Terminator 3: Rise of the Machines (2003)
17	13 Going on 30 (2004)	0.160118	Can't Hardly Wait (1998)
16	Titanic (1997)	0.153920	Cocktail (1988)
15	La La Land (2016)	0.148358	The Lookout (2007)
14	The Cabin in the Woods (2012)	0.143925	The Evil Dead (1981)
13	Clueless (1995)	0.141324	Escape from LA (1996)
12	Black Swan (2010)	0.116970	Sorority Boys (2002)
11	Interstellar (2014)	0.111184	Torque (2004)
10	Avatar (2009)	0.079484	Bad Boys (1995)

Figure Two: Dataframe contains the top 10 and last 10 easiest-to-predict movies and corresponding predictor and COD values

Question Two:

In this question, we fitted a multiple regression model for the 10 movies that are best and least well-predicted from question one with 3 extra categorical predictors which are gender identity, sibship status, and social viewing preference. The missing value of the gender identity column was handled using the most common value method as explained above. We also conducted the one-hot encoder for those columns since it can boost the model's performance and avoid the ordinal encoding assumptions. We recorded the new COD and plotted a scatter plot where the x-axis is the old COD value from the simple linear regression and the y-axis is the new COD value from the multiple linear regression.

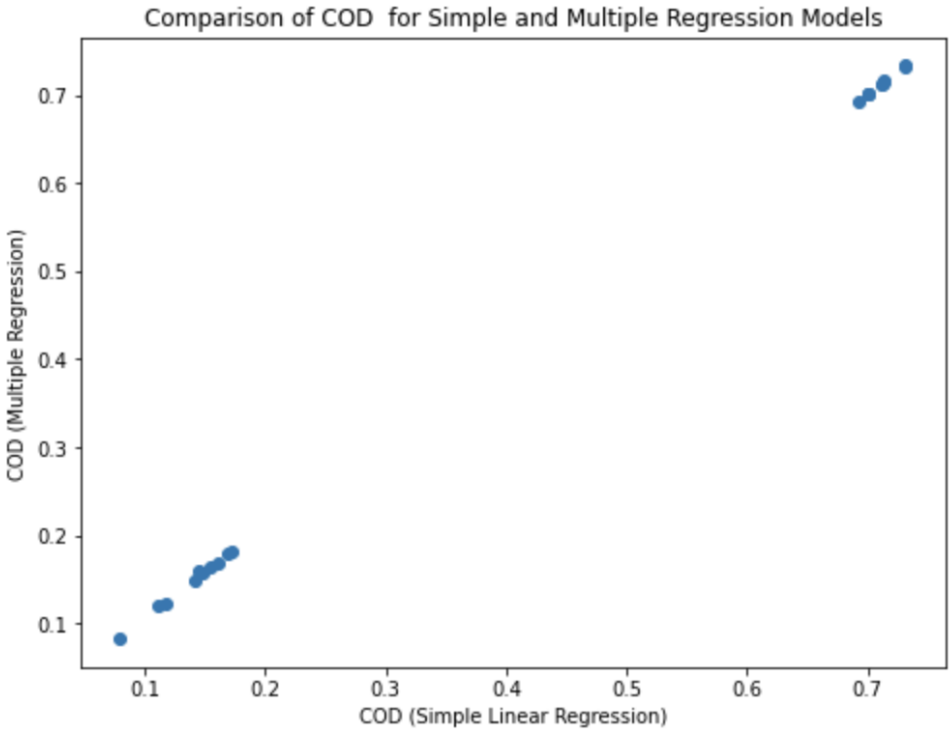


Figure Three: the comparison of COD values between simple and multiple regression models

Based on the scatter plot, the points are distributed in the 45-degree diagonal line. It means the improvement of adding three categorical predictors is very trivial.

Question Three:

In question three, we attempted to use the top 10 easiest-to-predict movies to predict the ratings of the 30 movies in the middle of the COD range under the regularization of ridge regression. The goal of ridge regression is to prevent overfitting by penalizing large coefficients in the regression equation. We also split the train/test set with an 80/20 ratio for the purposes of model evaluation and to prevent overfitting. In order to find the best value of the hyperparameter alpha for ridge regression, we first used the random search technique in the range of 1e-3 to 1e3 with 50 iterations and 5 folds cross-validation, which will give me a rough range of the best alpha. Then we used the grid search technique for a more detailed search of the optimal alpha based on the best alpha found in the random search method. After tuning the hyperparameter of ridge regression, we fitted the ridge regression model with the best alpha value. We calculated the training set RMSE and the testing set RMSE. We also extracted the best betas. All of these values have been recorded in a dataframe. Due to the length of the dataframe, we won't be able to attach it to the report. Feel free to check it out in the attached coding part. Based on the result of the dataframe, the betas shrank to a very small magnitude but none-zero. The training set RMSE and the testing set RMSE are very close to each other for most films, which means that there is no over-fitting except for a few films. The average testing RMSE is 0.38 which means the deviation from the predicted rating and actual rating is

0.38 overall. The best alpha has different values for different movies, ranging from the smallest 13 to the largest 104. The higher the alpha, the stronger the regularization.

Question Four:

The overall strategy here is similar to the last question, but we applied Lasso Regularization instead. Same here, we use the top 10 easiest-to-predict movies to predict the ratings of the 30 movies. The difference between RIDGE regression and LASSO is that, lasso regularization tends to reduce the number of the features, resulting in a sparse matrix in the weight of features(betas).

After splitting the train/test set with an 80/20 ratio, we try to find the best value of the hyperparameter alpha. Firstly we used the random search technique in the range of 1e-3 to 1e3 with 50 iterations and five-fold cross-validation, which will give us a rough range of the best alpha. Then we used the grid search technique for a more detailed search of the optimal alpha.

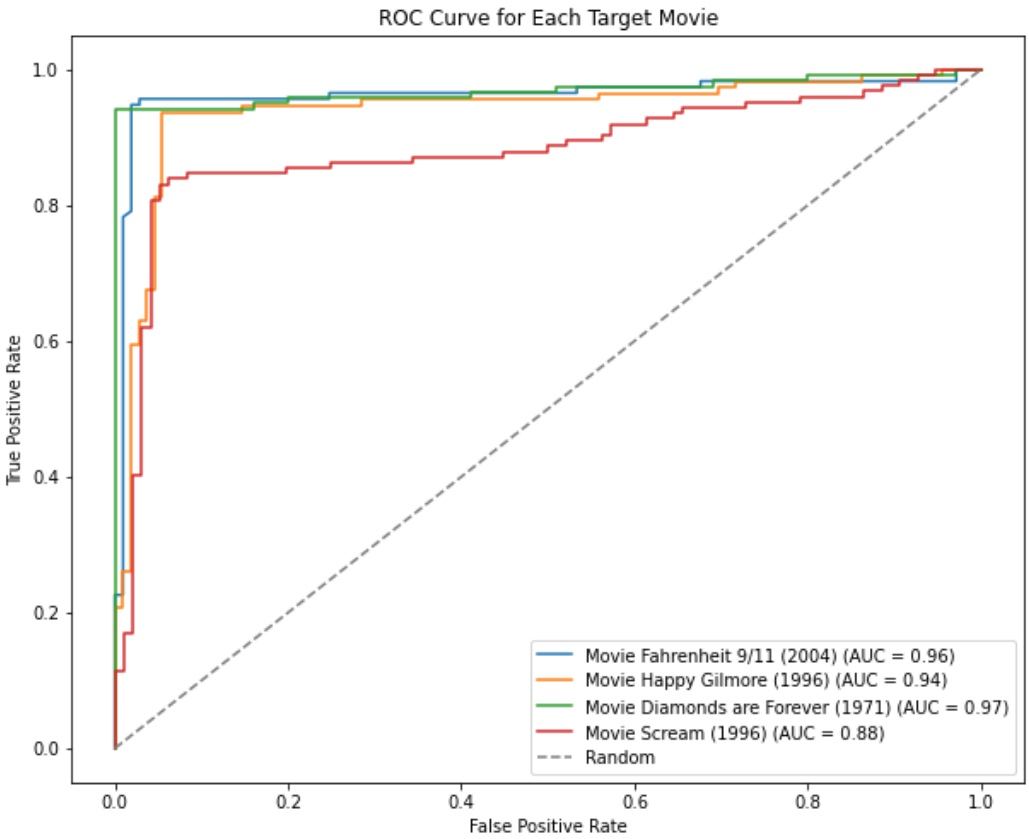
After tuning the hyperparameter of lasso regression, we fitted the lasso regression model with the best alpha value. We calculated the RMSE of the training set and the testing set. The RMSE, best alpha and beta values have been recorded in a dataframe, which is in the coding part attached after the report.

According to the values, the average testing RMSE is 0.385. And the alphas are much smaller here(0.001 to 0.021) compared to the alpha in the ridge regression. One possible reason we assume is that L1 regularization has a stronger power since it tends to "zerofy" the betas, so even small alpha values can make a big difference. Approximately 50% of the betas are zero. This complies with the characteristics of lasso regularization.

Question Five:

	Beta	AUC
Fahrenheit 9/11 (2004)	[7.28782442953909]	0.961781
Happy Gilmore (1996)	[4.930742398208231]	0.939582
Diamonds are Forever (1971)	[7.0936476691888215]	0.968833
Scream (1996)	[4.567408089572134]	0.881300

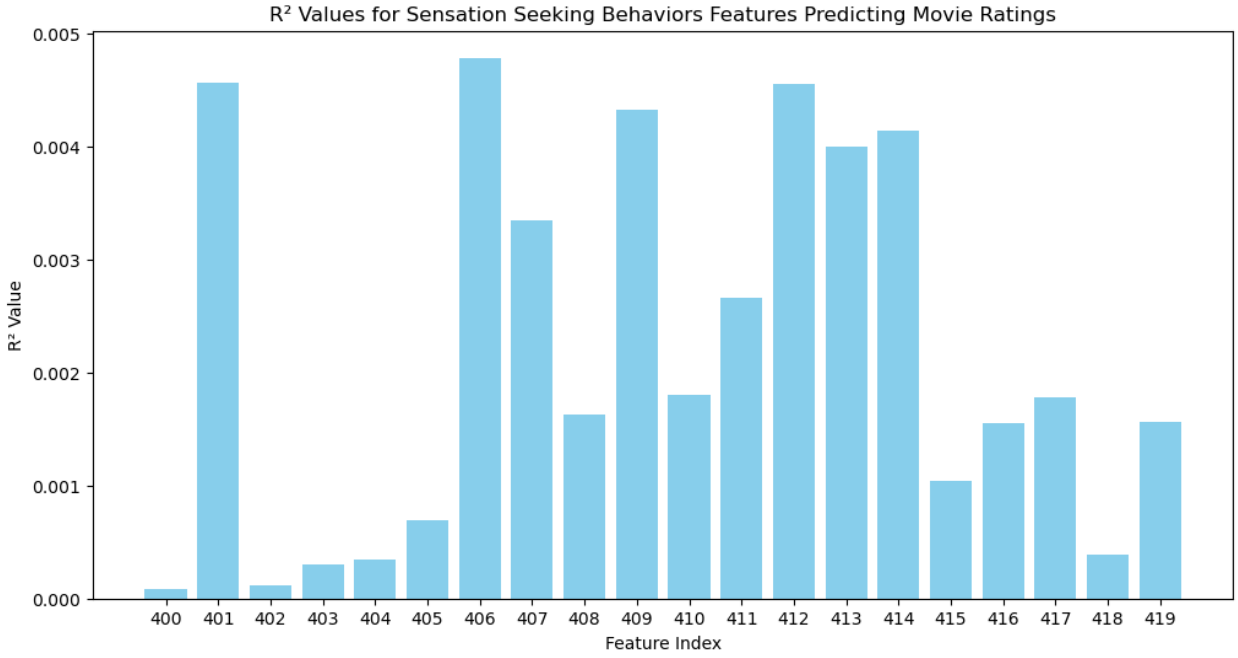
Movie Name	Model Accuracy with Cross-Validation					Mean
Fahrenheit 9/11 (2004)	0.960227	0.92000	0.942850	0.942850	0.954280	0.944045
Happy Gilmore (1996)	0.852272	0.885714	0.902857	0.857142	0.874285	0.874454
Diamonds are Forever (1971)	0.965909	0.960000	0.931428	0.977142	0.971428	0.961182
Scream (1996)	0.875000	0.811428	0.851428	0.914285	0.891428	0.868714



In this problem, before training the logistic regression model, we have found that all of the ratings from one user are missing, so we decide to remove that row because it is meaningless.
Then for each of the target movies, we first do a media split of ratings to achieve the Y labels. Secondly, we conduct five-cross-validation to avoid overfitting, and get the accuracy from each section. We put them into one table as shown above and calculate the mean accuracy. After that, we train the logistic regression model, obtain the ROC and measure the AUC.
From the above data and graph, we can conclude that the model is generally performing well in terms of correctly classifying outcomes, since accuracy(0.86~0.94 here) measures the proportion of true results (both true positives and true negatives) among the total number of cases examined. Plus that we use cross validation, it ensures that the stability is good. Based on the AUC values (0.88~0.96 here), the model exhibits good discriminative ability.When both AUC and accuracy are high, it suggests that the models are both stable and robust in terms of predictive performance.

Extra Credit:
In this part, we want to use “Columns 401-420: These columns contain self-assessments on sensation seeking behaviors (1-5)” as features to predict the average rating of each movie and compare which feature has the highest predicting ability. We selected COD as the evaluation

metric since we want to explore how much these features can interpret and account for the result ratings. And the graph below is the result:



This result makes sense since:

"Top 5 Predictors:
Have you ever been rock climbing?
I enjoy rollercoasters
Have you ever parachuted?
I enjoy watching horror movies
I had a sheltered upbringing

Last 5 Predictors:
I enjoy driving fast
Have you ever bungee-jumped?
I enjoy impulse shopping
I sometimes go out on weeknights even if I have work to do
Have you gambled or bet for money?"

From the above detailed outcome, we can see the top 5 predictors are more radical and are more prone to show the tendency towards stimulation of viewers, this means they probably have a high correlation with the ratings, while the last 5 predictors are more common and do not lead to radical sensation seeking that much, so they are not that good to be taken as predictors compared to the previous five features.

Data analysis project 2:

Applying machine learning methods to movie ratings data

Introduction to Data Science (DS-GA1001)

Code by: Kunkun Hu(kh3492@nyu.edu), Jiaran Peng(jp7238@nyu.edu)

Date: 11-13-23

In [1]:
```
# importing necessary libraries
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.metrics import mean_squared_error, accuracy_score,make_scorer
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
```

In [2]:
```
# importing df
df = pd.read_csv('movieReplicationSet.csv')
df.head()
```

Out[2]:

	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 th Righ One I (2008
0	NaN	NaN	4.0	NaN	3.0	NaN	NaN	NaN	Na
1	NaN	NaN	1.5	NaN	NaN	NaN	NaN	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	NaN	NaN	2.0	NaN	3.0	NaN	NaN	NaN	Na
4	NaN	NaN	3.5	NaN	0.5	NaN	0.5	1.0	Na

5 rows × 477 columns

Missing Data Handling

In [3]:
```
(df.iloc[:,:400].isna().sum(axis = 1)==400).sum()
```

Out[3]: 1

In [4]:
```
column_mean = df.iloc[:,:400].mean()
row_mean = df.iloc[:,:400].mean(axis = 1)
row_mean.isna()
```

Out[4]:
```
0       False
1       False
2       False
3       False
4       False
        ...
1092    False
1093    False
1094    False
1095    False
1096    False
Length: 1097, dtype: bool
```

In [5]:
```
movies_df = df.iloc[:,:400].copy()
def half_half_imputing(df):
    column_mean = df.mean()
    row_mean = df.mean(axis = 1)
    df_na = df.isna()
    for i in range(df.shape[0]):
        for j in df.columns:
            if df_na.loc[i,j]:
                if pd.isnull(row_mean[i]):
                    df.loc[i,j] = column_mean[j]
                else:

                    df.loc[i,j] = 0.5*row_mean[i]+0.5*column_mean[j]


half_half_imputing(movies_df)
movies_df
```

Out[5]:

	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)	
0	2.447086	2.381992	4.000000	2.725235	3.000000	2.670257	2.554121	2
1	2.439294	2.374200	1.500000	2.717443	2.752945	2.662464	2.546329	2
2	2.733065	2.667971	3.234118	3.011214	3.046716	2.956236	2.840100	
3	2.282975	2.217880	2.000000	2.561123	3.000000	2.506145	2.390009	2
4	2.209132	2.144038	3.500000	2.487281	0.500000	2.432303	0.500000	1
...	...	...	...	...	...	...	...	
1092	2.675658	2.610563	3.176711	2.953806	3.500000	2.898828	2.782692	3
1093	3.000000	4.000000	3.413546	3.190641	4.000000	4.000000	2.500000	3
1094	2.641923	2.576828	3.142976	2.920071	2.955574	2.865093	2.748957	3
1095	2.770970	2.705876	3.272023	3.049119	3.084621	2.994141	2.878005	
1096	2.512595	2.447500	4.000000	2.790743	2.500000	2.735765	2.619629	3

1097 rows × 400 columns

In [6]:
```
movies_df.isna().sum().sum()
```

Out[6]:
```
0
```

Question One: For each of the 400 movies, use a simple linear regression model to predict the ratings. Use the ratings of the *other* 399 movies in the dataset to predict the ratings of each movie (that means you'll have to build 399 models for each of the 400 movies). For each of the 400 movies, find the movie that predicts ratings the best. Then report the average COD of those 400 simple linear regression models. Please include a histogram of these 400 COD values and a table with the 10 movies that are most easily predicted from the ratings of a single other movie and the 10 movies that are hardest to predict from the ratings of a single other movie (and their associated COD values, as well as which movie ratings are the best predictor, so this table should have 3 columns).

In [7]:
```
columns = movies_df.columns
result_df = pd.DataFrame(columns = ["best_COD","best_predictor"])
for i in range(400):
    response = np.array(movies_df.iloc[:,i])
    best_r2 = 0
    best_movie = ""
    for j in range(i):
        predictor = np.array(movies_df.iloc[:,j]).reshape(-1,1)
        reg = LinearRegression().fit(predictor, response)
        y_hat = reg.predict(predictor)
        r2 = r2_score(response,y_hat)
        if r2>best_r2:
            best_r2 = r2
            best_movie = columns[j]
    for j in range(i+1,400):
        predictor = np.array(movies_df.iloc[:,j]).reshape(-1,1)
        reg = LinearRegression().fit(predictor, response)
        y_hat = reg.predict(predictor)
        r2 = r2_score(response,y_hat)
        if r2>best_r2:
            best_r2 = r2
            best_movie = columns[j]
    new_row_data = {"best_COD": best_r2, "best_predictor": best_movie}
    result_df.loc[columns[i]] = new_row_data
result_df.head()
```

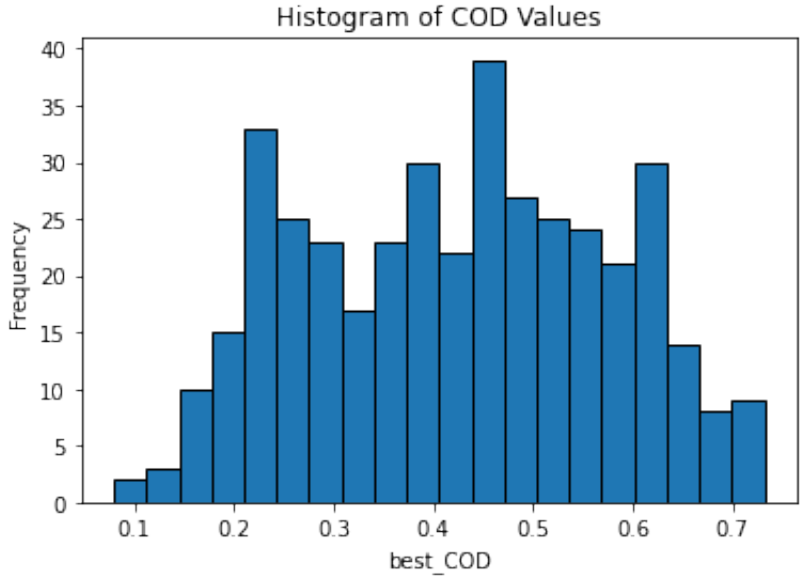
Out[7]:

	best_COD	best_predictor
The Life of David Gale (2003)	0.567937	The King of Marvin Gardens (1972)
Wing Commander (1999)	0.561052	From Hell (2001)
Django Unchained (2012)	0.232041	The Life of David Gale (2003)
Alien (1979)	0.329566	Aliens (1986)
Indiana Jones and the Last Crusade (1989)	0.374484	Indiana Jones and the Temple of Doom (1984)

In [8]:
```
result_df["best_COD"].mean()
```

Out[8]:
```
0.4237994048248016
```

In [9]:
```
plt.hist(result_df['best_COD'],edgecolor='black',bins = 20)
plt.title('Histogram of COD Values')
plt.xlabel('best_COD')
plt.ylabel('Frequency')
plt.show()
```



In [10]:
```
top_last_10 = pd.concat([result_df.nlargest(10,"best_COD"), result_df.nsmall
```

In [11]:
```
top_last_10=top_last_10.reset_index().rename(columns={'index': 'movie'})
top_last_10.sort_values("best_COD", ascending=False)
```

Out[11]:

	movie	best_COD	best_predictor
0	Erik the Viking (1989)	0.731789	I.Q. (1994)
1	I.Q. (1994)	0.731789	Erik the Viking (1989)
2	The Lookout (2007)	0.713793	Patton (1970)
3	Patton (1970)	0.713793	The Lookout (2007)
4	The Bandit (1996)	0.711540	Best Laid Plans (1999)
5	Best Laid Plans (1999)	0.711540	The Bandit (1996)
6	Congo (1995)	0.700822	The Straight Story (1999)
7	The Straight Story (1999)	0.700822	Congo (1995)
8	The Final Conflict (1981)	0.700437	The Lookout (2007)
9	Ran (1985)	0.692863	Heavy Traffic (1973)
19	Grown Ups 2 (2013)	0.171151	The Core (2003)
18	The Fast and the Furious (2001)	0.169000	Terminator 3: Rise of the Machines (2003)
17	13 Going on 30 (2004)	0.160118	Can't Hardly Wait (1998)
16	Titanic (1997)	0.153920	Cocktail (1988)
15	La La Land (2016)	0.148358	The Lookout (2007)
14	The Cabin in the Woods (2012)	0.143925	The Evil Dead (1981)
13	Clueless (1995)	0.141324	Escape from LA (1996)
12	Black Swan (2010)	0.116970	Sorority Boys (2002)
11	Interstellar (2014)	0.111184	Torque (2004)
10	Avatar (2009)	0.079484	Bad Boys (1995)

Question Two: For the 10 movies that are best and least well predicted from the ratings of a single other movie (so 20 in total), build multiple regression models that include gender identity (column 475), sibship status (column 476) and social viewing preferences (column 477) as additional predictors (in addition to the best predicting movie from question 1). Comment on how R^2 has changed relative to the answers in question 1. Please include a figure with a scatterplot where the old COD (for the simple linear regression models from the

previous question) is on the x-axis and the new R^2 (for the new multiple regression models) is on the y-axis.

疑问需不需要用one hot 来deal with categorical predictors

In [12]:
```
# there are NA in gender column.
df.iloc[:,[474,475,476]].isna().sum()
```

Out[12]:
```
Gender identity (1 = female; 2 = male; 3 = self-described)           24
Are you an only child? (1: Yes; 0: No; -1: Did not respond)           0
Movies are best enjoyed alone (1: Yes; 0: No; -1: Did not respond)    0
dtype: int64
```

In [13]:
```
df.iloc[:,474].value_counts()
```

Out[13]:
```
Gender identity (1 = female; 2 = male; 3 = self-described)
1.0    807
2.0    260
3.0      6
Name: count, dtype: int64
```

In [14]:
```
# since the gender column is not balanced and most are female. It makes sens
df.iloc[:,474]
df.iloc[:,474].fillna(df.iloc[:,474].mode().iloc[0], inplace=True)
```

In [15]:
```
encoder = OneHotEncoder()
categorical_1hot = encoder.fit_transform(df.iloc[:,[474,475,476]])
categorical_feartures=categorical_1hot.toarray()
categorical_feartures.shape
```

Out[15]:
```
(1097, 9)
```

In [16]:
```
top_last_10_new = pd.DataFrame(columns = ["COD_with_multiply_features"])
for movie, predictor in zip(top_last_10["movie"],top_last_10["best_predictor
    continuous_feature = np.array(movies_df[predictor]).reshape(-1,1)
    new_predictors = np.concatenate((continuous_feature,categorical_fearture
    response = np.array(movies_df[movie])
    reg = LinearRegression().fit(new_predictors, response)
    y_hat = reg.predict(new_predictors)
    r2 = r2_score(response,y_hat)
    top_last_10_new.loc[movie] = r2
top_last_10_new
```

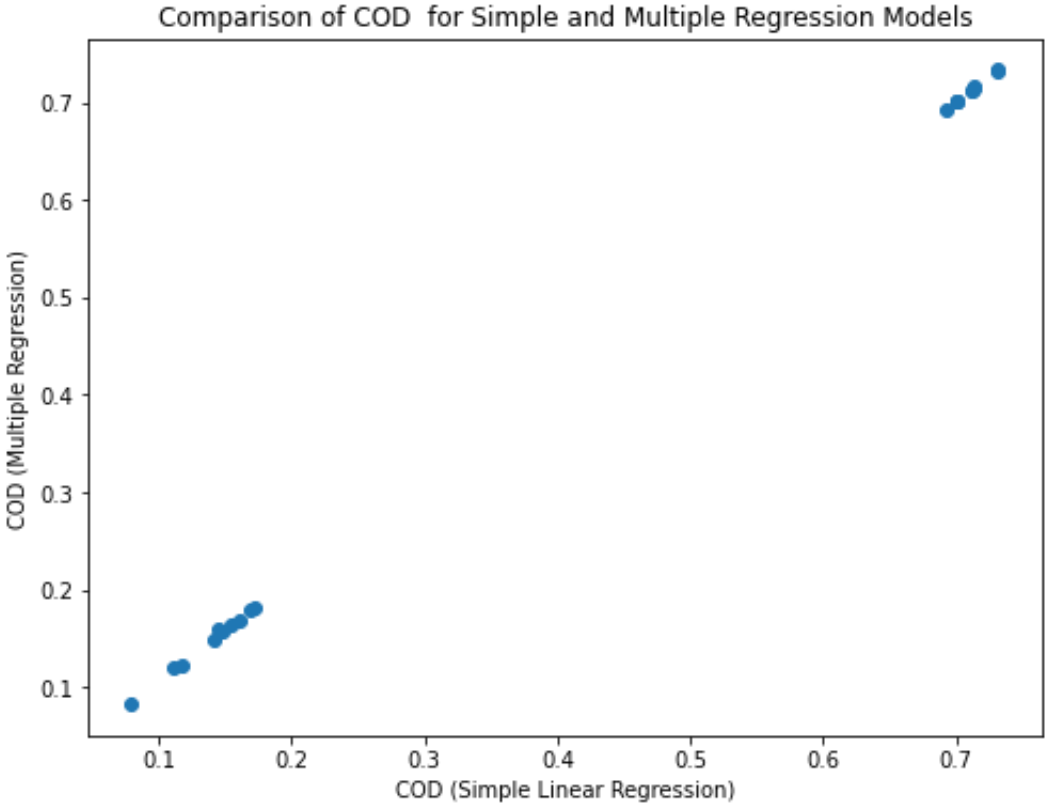
Out[16]:

	COD_with_multiply_features
Erik the Viking (1989)	0.733508
I.Q. (1994)	0.732317
The Lookout (2007)	0.716008
Patton (1970)	0.715004
The Bandit (1996)	0.713525
Best Laid Plans (1999)	0.712983
Congo (1995)	0.701177
The Straight Story (1999)	0.702577
The Final Conflict (1981)	0.702597
Ran (1985)	0.693241
Avatar (2009)	0.082782
Interstellar (2014)	0.118971
Black Swan (2010)	0.123043
Clueless (1995)	0.147923
The Cabin in the Woods (2012)	0.159037
La La Land (2016)	0.156376
Titanic (1997)	0.162932
13 Going on 30 (2004)	0.168076
The Fast and the Furious (2001)	0.178218
Grown Ups 2 (2013)	0.181741

In [17]:
```
# Didn't use one hot encoder
#top_last_10_new = pd.DataFrame(columns = ["COD_with_multiply_features"])
#for movie, predictor in zip(top_last_10["movie"],top_last_10["best_predicto
#    new_predictors = np.array(pd.concat([movies_df[predictor],df.iloc[:,[47
#    response = np.array(movies_df[movie])
#    reg = LinearRegression().fit(new_predictors, response)
#    y_hat = reg.predict(new_predictors)
#    r2 = r2_score(response,y_hat)
#    top_last_10_new.loc[movie] = r2
#top_last_10_new
```

In [18]:
```
plt.figure(figsize=(8, 6))
```

```
plt.scatter(top_last_10['best_COD'], top_last_10_new['COD_with_multiply_feat
plt.title('Comparison of COD  for Simple and Multiple Regression Models')
plt.xlabel('COD (Simple Linear Regression)')
plt.ylabel('COD (Multiple Regression)')
plt.show()
```



Question Three: Pick 30 movies in the middle of the COD range, as identified by question 1 (that were not used in question 2). Now build a regularized regression model with the ratings from 10 other movies (picked randomly, or deliberately by you) as an input. Please use ridge regression, and make sure to do suitable hyperparameter tuning. Also make sure to report the RMSE for each of these 30 movies in a table, after doing an 80/20 train/test split. Comment on the hyperparameters you use and betas you find by doing so.

In [19]:
```
# choose the top 10 easiest predict movies as predictors
# using random search and then grid search for hyperparameter tuning
new_movies = result_df.sort_values(by='best_COD', ascending=False).index[185
predictors = result_df.sort_values(by='best_COD', ascending=False).index[:10
predictors = np.array(movies_df.loc[:,predictors])
result_ridge = pd.DataFrame(columns = ["best_alpha","rmse_train","rmse_test"
for movie in new_movies:
    response = np.array(movies_df[movie])
```

```
    X_train, X_test, y_train, y_test = train_test_split(predictors, response
    ridge_model = Ridge()
    param_grid = {'alpha': np.logspace(-3, 3, 100)}
    random_search = RandomizedSearchCV(ridge_model, param_grid, scoring='neg
    random_search.fit(X_train, y_train)
    best_alpha_random = random_search.best_params_['alpha']
    grid_param_grid = {'alpha': np.linspace(best_alpha_random / 2, best_alph
    grid_search = GridSearchCV(ridge_model, grid_param_grid, scoring='neg_me
    grid_search.fit(X_train, y_train)
    best_alpha_grid = grid_search.best_params_['alpha']
    final_model = Ridge(alpha=best_alpha_grid)
    final_model.fit(X_train, y_train)
    coefficients = final_model.coef_.round(4)
    y_pred_test = final_model.predict(X_test)
    y_pred_train = final_model.predict(X_train)
    rmse_train = np.sqrt(mean_squared_error(y_train, y_pred_train))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
    result_ridge.loc[movie] = [best_alpha_grid,rmse_train,rmse_test,coeffici

result_ridge
```

Out[19]:

	best_alpha	rmse_train	rmse_test	betas
The Prestige (2006)	85.382193	0.331824	0.384023	[0.0909, 0.1558, 0.0615, 0.08, 0.0776, 0.0684,...
Man on Fire (2004)	52.523242	0.375016	0.284170	[0.0808, 0.2036, 0.157, 0.06, 0.1013, 0.1967, ...
Reservoir Dogs (1992)	91.785857	0.324480	0.358640	[0.0629, 0.0908, 0.149, 0.0759, 0.0978, 0.0736...
Just Married (2003)	42.494919	0.360222	0.373650	[0.1249, 0.1508, 0.096, 0.0785, 0.0375, 0.0685...
The Mummy (1999)	91.785857	0.542210	0.508630	[0.0795, 0.0434, 0.0284, 0.1064, 0.0702, 0.111...
The Mummy Returns (2001)	85.382193	0.475527	0.484910	[0.0551, 0.0955, 0.0578, 0.0865, 0.1449, 0.132...
Equilibrium (2002)	104.404966	0.314527	0.300639	[0.0636, 0.0721, 0.0418, 0.0651,

				0.0499, 0.075...
Let the Right One In (2008)	39.307800	0.311621	0.270163	[0.0121, 0.0595, 0.1386, 0.0705, 0.0628, 0.059...
The Good the Bad and the Ugly (1966)	85.382193	0.341563	0.438675	[0.0511, 0.0867, 0.0767, 0.1312, 0.0888, 0.129...
The Poseidon Adventure (1972)	32.933562	0.288715	0.459654	[0.0291, 0.115, 0.119, 0.0466, 0.0471, 0.098, ...
You're Next (2011)	13.599204	0.313745	0.325984	[0.0552, -0.0374, 0.1921, 0.2598, 0.2763, 0.07...
The Rock (1996)	20.669582	0.341368	0.290794	[0.1045, 0.046, 0.0295, 0.2325, 0.132, 0.2645,...
The Green Mile (1999)	95.939699	0.349459	0.294711	[0.0615, 0.0464, 0.1188, 0.0848, 0.0907, 0.043...
Men in Black II (2002)	95.939699	0.516305	0.484604	[0.0954, 0.1172, 0.0792, 0.1314, 0.0943, 0.124...
Men in Black (1997)	104.404966	0.517202	0.519387	[0.1016, 0.0794, 0.1056, 0.1507, 0.1215, 0.105...
The Blue Lagoon (1980)	63.121915	0.347630	0.318726	[0.0531, 0.0426, 0.0854, 0.1709, 0.1152, 0.049...
Uptown Girls (2003)	36.120681	0.385274	0.402160	[0.1446, 0.0909, 0.1907, 0.0916, 0.0609, 0.111...
The Machinist (2004)	68.691496	0.344919	0.270360	[0.0385, 0.084, 0.1181, 0.0597, 0.0145, 0.1121...
The Evil Dead (1981)	63.121915	0.313032	0.400959	[0.0823, 0.1518, 0.0929, 0.0959, 0.0755, 0.125...
Knight and Day (2010)	15.999064	0.344782	0.420162	[0.0572, 0.145, -0.0099, 0.1447, 0.3271, 0.134...
				[-0.0481,

28 Days Later (2002)	17.015408	0.335579	0.410027	-0.0457, 0.3052, 0.1429, 0.1457, 0.0...
Dances with Wolves (1990)	63.121915	0.300021	0.501081	[0.0164, 0.1494, 0.1715, 0.1188, 0.0406, 0.021...
Blues Brothers 2000 (1998)	68.691496	0.365451	0.320864	[0.07, 0.0738, 0.0882, 0.1157, 0.1277, 0.0276,...
Twister (1996)	68.691496	0.331800	0.344637	[0.0555, 0.0913, 0.1226, 0.1192, 0.0946, 0.159...
The Big Lebowski (1998)	25.861907	0.353212	0.305075	[-0.0025, -0.0082, 0.2268, 0.1666, -0.0118, 0....
Goodfellas (1990)	17.015408	0.339183	0.354098	[0.0958, 0.193, 0.3124, -0.1361, 0.1336, 0.085...
Austin Powers: The Spy Who Shagged Me (1999)	30.055730	0.548150	0.511058	[0.0999, 0.2405, 0.2992, 0.1385, 0.0753, 0.194...
Austin Powers in Goldmember (2002)	57.552335	0.537839	0.480314	[0.101, 0.0413, 0.1634, 0.149, 0.0667, 0.1222,...
Crossroads (2002)	68.691496	0.356536	0.302674	[0.0496, 0.0625, 0.1188, 0.1061, 0.0735, 0.173...
Gone in Sixty Seconds (2000)	56.187654	0.337778	0.309880	[0.0948, 0.0142, 0.1388, 0.0882, 0.1471, 0.112...

In [20]:
```
result_ridge["rmse_test"].mean()
```

Out[20]:
```
0.38102365820437334
```

Question Four: Repeat question 3) with LASSO regression. Again, make sure to comment on the hyperparameters you use and betas you find by doing so.

In [21]:
```
result_lasso = pd.DataFrame(columns = ["best_alpha","rmse_train","rmse_test"
for movie in new_movies:
    response = np.array(movies_df[movie])
    X_train, X_test, y_train, y_test = train_test_split(predictors, response
    lasso_model = Lasso()
    param_grid = {'alpha': np.logspace(-3, 3, 100)}
    random_search = RandomizedSearchCV(lasso_model, param_grid, scoring='neg
    random_search.fit(X_train, y_train)
    best_alpha_random = random_search.best_params_['alpha']
    grid_param_grid = {'alpha': np.linspace(best_alpha_random / 2, best_alph
    grid_search = GridSearchCV(lasso_model, grid_param_grid, scoring='neg_me
    grid_search.fit(X_train, y_train)
    best_alpha_grid = grid_search.best_params_['alpha']
    final_model = Lasso(alpha=best_alpha_grid)
    final_model.fit(X_train, y_train)
    coefficients = final_model.coef_.round(4)
    y_pred_test = final_model.predict(X_test)
    y_pred_train = final_model.predict(X_train)
    rmse_train = np.sqrt(mean_squared_error(y_train, y_pred_train))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))
    result_lasso.loc[movie] = [best_alpha_grid,rmse_train,rmse_test,coeffici

result_lasso
```

Out[21]:

	best_alpha	rmse_train	rmse_test	betas
The Prestige (2006)	0.003419	0.326744	0.375762	[0.0, 0.3692, 0.0, 0.1168, 0.073, 0.0296, 0.0,...
Man on Fire (2004)	0.006459	0.370516	0.297844	[0.0, 0.3569, 0.2256, 0.0, 0.0, 0.3621, 0.0, 0...
Reservoir Dogs (1992)	0.005974	0.319747	0.355966	[0.0, 0.0781, 0.3147, 0.0, 0.1515, 0.0103, 0.0...
Just Married (2003)	0.001211	0.352186	0.395548	[0.0473, 0.244, 0.0809, 0.0, 0.0, 0.1171, -0.3...
The Mummy (1999)	0.011031	0.540375	0.523733	[0.0, 0.0, 0.0, 0.0793, 0.0, 0.1668, 0.1799, 0...
The Mummy Returns (2001)	0.014921	0.474611	0.492844	[0.0, 0.0849, 0.0, 0.0, 0.2467,

				0.1863, 0.1064...
Equilibrium (2002)	0.009081	0.311471	0.305581	[0.0, 0.1054, 0.0, 0.0086, 0.0, 0.1092, 0.0102...
Let the Right One In (2008)	0.003419	0.310340	0.273183	[-0.0, 0.0, 0.2232, 0.0, 0.0166, 0.0463, 0.179...
The Good the Bad and the Ugly (1966)	0.006459	0.337985	0.432784	[0.0, 0.1037, 0.0, 0.2799, 0.0207, 0.2248, 0.0...
The Poseidon Adventure (1972)	0.003931	0.286984	0.468504	[0.0, 0.0861, 0.1542, 0.0, 0.0, 0.1099, 0.19, ...
You're Next (2011)	0.006459	0.314145	0.323213	[0.0, 0.0, 0.1809, 0.3271, 0.3211, 0.0, 0.0, 0...
The Rock (1996)	0.005196	0.341360	0.291933	[0.088, 0.0, 0.0, 0.3098, 0.0642, 0.3269, -0.0...
The Green Mile (1999)	0.007257	0.345440	0.309624	[0.0, 0.0, 0.2244, 0.0, 0.1495, 0.0, 0.0, 0.02...
Men in Black II (2002)	0.016040	0.516004	0.486820	[0.0117, 0.1887, 0.0, 0.208, 0.0062, 0.1575, 0...
Men in Black (1997)	0.020977	0.516686	0.522842	[0.149, 0.0, 0.0, 0.3173, 0.1533, 0.0437, 0.0,...
The Blue Lagoon (1980)	0.007898	0.343692	0.324103	[0.0, 0.0, 0.0, 0.3682, 0.1486, 0.0, 0.0, 0.20...
Uptown Girls (2003)	0.005618	0.382789	0.396005	[0.1565, 0.0143, 0.3444, 0.0, 0.0, 0.1655, 0.0...
The Machinist (2004)	0.004249	0.341201	0.282866	[-0.0, 0.0664, 0.1684, 0.0, -0.0, 0.1484, -0.0...
				[0.0, 0.3005, 0.0732, 0.0939,

The Evil Dead (1981)	0.006459	0.310297	0.395598	0.0, 0.2045, 0.0...
Knight and Day (2010)	0.000737	0.341657	0.428029	[0.0, 0.2048, -0.0534, 0.1389, 0.4976, 0.1084,...
28 Days Later (2002)	0.005974	0.336720	0.401763	[-0.0, -0.0, 0.37, 0.1034, 0.1349, 0.0, 0.0461...
Dances with Wolves (1990)	0.006943	0.295881	0.509718	[-0.0, 0.2068, 0.3047, 0.1107, 0.0, 0.0, 0.0, ...
Blues Brothers 2000 (1998)	0.005618	0.361248	0.338839	[0.0, 0.0155, 0.0764, 0.1474, 0.2014, -0.0, -0...
Twister (1996)	0.009817	0.329490	0.355008	[0.0, 0.0855, 0.1997, 0.1275, 0.0006, 0.3359, ...
The Big Lebowski (1998)	0.006459	0.352897	0.303314	[0.0, 0.0, 0.2917, 0.1471, 0.0, 0.2356, 0.0, 0...
Goodfellas (1990)	0.002274	0.337039	0.364302	[0.0174, 0.2119, 0.4437, -0.2351, 0.1308, 0.08...
Austin Powers: The Spy Who Shagged Me (1999)	0.007257	0.546373	0.511936	[0.0, 0.3269, 0.4741, 0.0084, 0.0, 0.288, 0.0,...
Austin Powers in Goldmember (2002)	0.012004	0.537110	0.484958	[0.0693, 0.0, 0.2211, 0.1294, 0.0, 0.1534, 0.0...
Crossroads (2002)	0.003931	0.351048	0.312916	[0.0, 0.0, 0.2289, 0.0425, 0.0, 0.405, 0.0, 0....
Gone in Sixty Seconds (2000)	0.005618	0.335989	0.308974	[0.0219, -0.0, 0.2097, 0.0, 0.2252, 0.1053, -0...

Question Five: Compute the average movie enjoyment for each user (using only real, non-imputed data). Use these averages as the predictor variable X in a logistic regression model. Sort the movies order of increasing rating (also using only real, non-imputed data). Now pick the 4 movies in the middle of the score range as your target movie. For each of them, do a media split (now using the imputed data) of ratings to code movies above the median rating with the Y label 1 (= enjoyed) and movies below the median with the label 0 (= not enjoyed). For each of these movies, build a logistic regression model (using X to predict Y), show figures with the outcomes and report the betas as well as the AUC values. Comment on the quality of your models. Make sure to use cross-validation methods to avoid overfitting

In [22]:
```
df_with_none = pd.read_csv('movieReplicationSet.csv')
df_with_none_movie = df_with_none.iloc[:,:400]
df_with_none_movie[df_with_none_movie.mean(axis = 1).isna()]
```

Out[22]:

	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 R Or (20
896	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

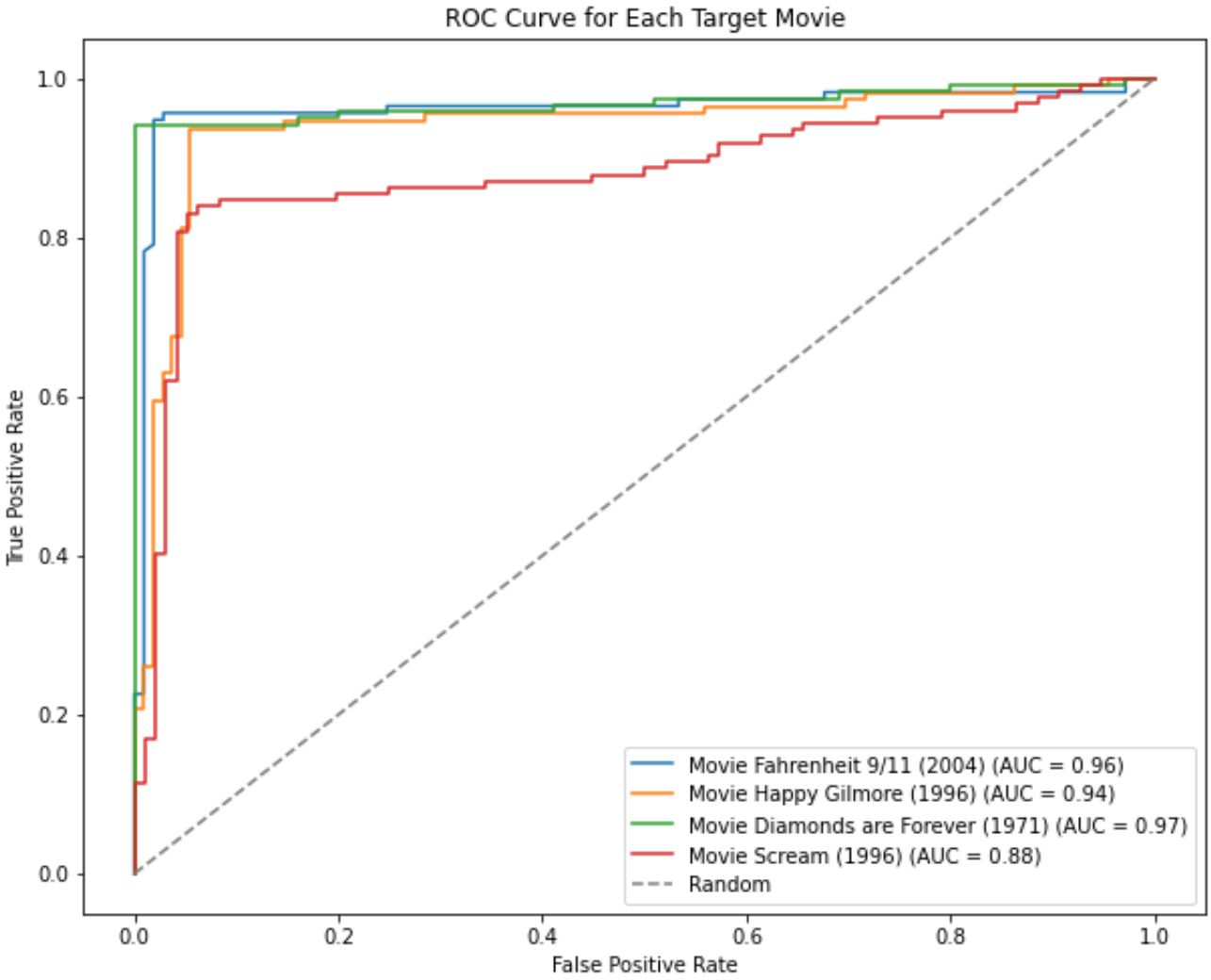
1 rows × 400 columns

In [23]:
```
# that user didn't watch any movie and we can't compute the average enjoymer
index_to_drop = df_with_none_movie[df_with_none_movie.mean(axis = 1).isna()]
df_with_none_movie = df_with_none.iloc[:,:400].drop(index_to_drop)
```

In [24]:
```
movies_df_drop_na = movies_df.drop(index_to_drop)
X=np.array(df_with_none_movie.mean(axis = 1)).reshape(-1,1)
movie_mean_rating = df_with_none_movie.mean().sort_values()
target_movies = movie_mean_rating.index[198:202]
result_df_logi = pd.DataFrame(columns=['Beta', 'AUC'])

plt.figure(figsize=(10, 8))
for movie in target_movies:

    median_rating = movies_df_drop_na[movie].median()
    Y = np.array((movies_df_drop_na[movie] > median_rating).astype(int))
```

```
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
    model = LogisticRegression()
    cv_scores = cross_val_score(model, X_train, Y_train, cv=5, scoring='accu
    mean_auc = cv_scores.mean()
    print(f"mean accuracy:{mean_auc}     cv accuracy:{cv_scores}")
    model.fit(X_train, Y_train)
    Y_pred_proba = model.predict_proba(X_test)[:, 1]


    auc_score = roc_auc_score(Y_test, Y_pred_proba)

    # Plot ROC curve
    fpr, tpr, _ = roc_curve(Y_test, Y_pred_proba)
    plt.plot(fpr, tpr, label=f'Movie {movie} (AUC = {auc_score:.2f})')

    result_df_logi.loc[movie] = [model.coef_[0],auc_score]

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Each Target Movie')
plt.legend()
plt.show()

result_df_logi
```

```
mean accuracy:0.9440454545454546     cv accuracy:[0.96022727 0.92       0.94
285714 0.94285714 0.95428571]
mean accuracy:0.8744545454545454     cv accuracy:[0.85227273 0.88571429 0.90
285714 0.85714286 0.87428571]
mean accuracy:0.9611818181818181     cv accuracy:[0.96590909 0.96       0.93
142857 0.97714286 0.97142857]
mean accuracy:0.8687142857142856     cv accuracy:[0.875      0.81142857 0.85
142857 0.91428571 0.89142857]
```



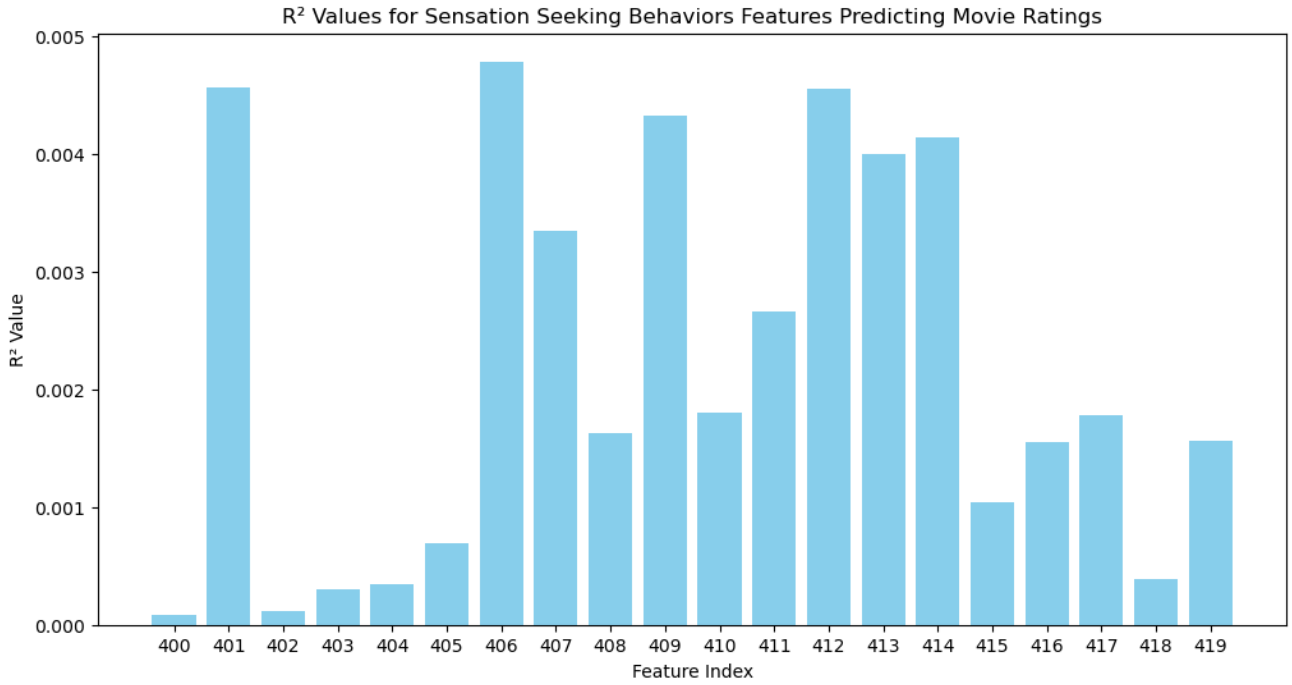
Out[24]:

	Beta	AUC
Fahrenheit 9/11 (2004)	[7.28782442953909]	0.961781
Happy Gilmore (1996)	[4.930742398208231]	0.939582
Diamonds are Forever (1971)	[7.0936476691888215]	0.968833
Scream (1996)	[4.567408089572134]	0.881300

In [ ]:

questions above [for an additional 5% of the grade score].

In [80]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

Y = imputed_df.iloc[:, :400].mean(axis=1)

feature_indices = list(range(400, 420))
r2_values = []

for feature_index in feature_indices:
    X = imputed_df.iloc[:, feature_index].values.reshape(-1, 1)

    # Linear Regression Model
    model = LinearRegression().fit(X, Y)
    Y_pred = model.predict(X)
    r2 = r2_score(Y, Y_pred)

    r2_values.append(r2)

plt.figure(figsize=(12, 6))
plt.bar(feature_indices, r2_values, color='skyblue')
plt.xlabel('Feature Index')
plt.ylabel('R² Value')
plt.title('R² Values for Sensation Seeking Behaviors Features Predicting Mov
plt.xticks(feature_indices)
plt.show()

top_5_indices = sorted(range(len(r2_values)), key=lambda i: r2_values[i], re
top_5_feature_names = [imputed_df.columns[400+i] for i in top_5_indices]
print("Top 5 Predictors:")
for feature_name in top_5_feature_names:
    print(feature_name)

last_5_indices = sorted(range(len(r2_values)), key=lambda i: r2_values[i], r
last_5_feature_names = [imputed_df.columns[400+i] for i in last_5_indices]
print("\nLast 5 Predictors:")
for feature_name in last_5_feature_names:
    print(feature_name)
```




```
Top 5 Predictors:
Have you ever been rock climbing?
I enjoy rollercoasters
Have you ever parachuted?
I enjoy watching horror movies
I had a sheltered upbringing

Last 5 Predictors:
I enjoy driving fast
Have you ever bungee-jumped?
I enjoy impulse shopping
I sometimes go out on weeknights even if I have work to do
Have you gambled or bet for money?
```

In [ ]: