In [31]:
```
import pandas as pd
```

In [32]:
```
# Load the dataset
data = pd.read_csv('movieReplicationSet.csv')
data
```

Out[32]:

	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)	...
0	NaN	NaN	4.0	NaN	3.0	NaN	NaN	NaN	NaN	NaN	...

In [33]:
```
# Isolating the movie ratings section of the dataset (first 400 columns)
movie_ratings = data.iloc[:, :400]

# Calculate the mean for each movie (column mean) and for each user (row
column_means = movie_ratings.mean(axis=0, skipna=True)
row_means = movie_ratings.mean(axis=1, skipna=True)

# Impute missing values using the 50/50 blend of row mean and column mea
for row_index, row in movie_ratings.iterrows():
    for col_index in row[row.isna()].index:
        user_mean = row_means[row_index]
        movie_mean = column_means[col_index]
        imputed_value = (user_mean + movie_mean) / 2
        movie_ratings.at[row_index, col_index] = imputed_value

# Check the first few rows after imputation
movie_ratings.head()
```

Out[33]:

	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)
0	2.447086	2.381992	4.000000	2.725235	3.000000	2.670257	2.554121	2.821232	2.619604
1	2.439294	2.374200	1.500000	2.717443	2.752945	2.662464	2.546329	2.813440	2.611812
2	2.733065	2.667971	3.234118	3.011214	3.046716	2.956236	2.840100	3.107211	2.905583
3	2.282975	2.217880	2.000000	2.561123	3.000000	2.506145	2.390009	2.657120	2.455492
4	2.209132	2.144038	3.500000	2.487281	0.500000	2.432303	0.500000	1.000000	2.381650

5 rows × 400 columns

In [34]:
```
# Isolating the remaining columns (401-477) for imputation
remaining_data = data.iloc[:, 400:]

# Calculate the mean for each of these columns, excluding NaNs
column_means_remaining = remaining_data.mean(axis=0, skipna=True)

# Impute missing values in these columns using the column means
remaining_data_imputed = remaining_data.fillna(column_means_remaining)

# Check the first few rows after imputation for these columns
remaining_data_imputed.head()
```

Out[34]:

	I enjoy driving fast	I enjoy rollercoasters	Have you ever bungee-jumped?	I enjoy impulse shopping	I sometimes go out on weeknights even if I have work to do	I enjoy doing things without too much planning	Have you ever been rock climbing?	I enjoy being in large loud crowds like the Times Square Ball Drop on New Years Eve	I enjoy going to large music or dance festivals
0	5.0	5.0	2.0	5.0	1.0	2.0	3.0	1.0	4.0
1	4.0	5.0	2.0	4.0	2.0	1.0	1.0	2.0	4.0
2	4.0	4.0	1.0	2.0	2.0	2.0	1.0	3.0	4.0
3	5.0	5.0	2.0	5.0	4.0	2.0	4.0	4.0	5.0
4	4.0	1.0	3.0	3.0	2.0	3.0	3.0	1.0	3.0

5 rows × 77 columns

In [35]:
```
# average movie enjoyment per user (considering only the first 400 colum
user_average_ratings = data.iloc[:, :400].mean(axis=1)  # Considering on

# Displaying the first few average ratings
user_average_ratings.head()
```

Out[35]:
```
0    2.742857
1    2.727273
2    3.314815
3    2.414634
4    2.266949
dtype: float64
```

In [36]:
```
movie_average_ratings = data.iloc[:, :400].mean(axis=0)
# Sorting movies by their average ratings
sorted_movies = movie_average_ratings.sort_values()

# Selecting the 4 movies in the middle of the score range
middle_movies_count = len(sorted_movies) // 2
target_movies = sorted_movies[middle_movies_count - 2 : middle_movies_co
# Displaying the selected target movies
target_movies_index = target_movies.index
print(target_movies)
print(target_movies_index)
```

```
Fahrenheit 9/11 (2004)         2.578014
Happy Gilmore (1996)           2.581169
Diamonds are Forever (1971)    2.582677
Scream (1996)                  2.584270
dtype: float64
Index(['Fahrenheit 9/11 (2004)', 'Happy Gilmore (1996)',
       'Diamonds are Forever (1971)', 'Scream (1996)'],
      dtype='object')
```

In [37]:
```
# Performing a median split using the imputed data for the revised selec
median_split_labels_imputed = {}

for movie in target_movies_index:
    # Find the median rating for each movie using the imputed data
    median_rating_imputed = movie_ratings[movie].median()

    # Labeling movies above the median as 1 and below as 0
    labels_imputed = movie_ratings[movie].apply(lambda x: 1 if x >= medi
    median_split_labels_imputed[movie] = labels_imputed

# Creating a DataFrame for the median split labels using imputed data
median_split_df_imputed = pd.DataFrame(median_split_labels_imputed)

# Displaying the first few rows of the median split labels using imputed
median_split_df_imputed.head()
```

Out[37]:

	Fahrenheit 9/11 (2004)	Happy Gilmore (1996)	Diamonds are Forever (1971)	Scream (1996)
0	0	1	0	0
1	0	0	0	0
2	1	1	1	1
3	0	0	0	0
4	0	0	0	0

In [38]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score, roc_curve, auc
import numpy as np
import matplotlib.pyplot as plt

# Preparing the predictor variable (X) and initializing the dictionary to
X = user_average_ratings.values.reshape(-1, 1)  # Reshaping for compatib
model_results = {}
```

In [39]:
```
# Checking for NaN or infinite values in the predictor variable (X)
nan_in_X = np.isnan(X).any()
infinite_in_X = np.isinf(X).any()

# Checking for NaN or infinite values in the target variables (Y) for eac
nan_in_Y = {movie: median_split_df_imputed[movie].isna().any() for movie
infinite_in_Y = {movie: np.isinf(median_split_df_imputed[movie]).any() fo

nan_in_X, infinite_in_X, nan_in_Y, infinite_in_Y
```

Out[39]:
```
(True,
 False,
 {'Fahrenheit 9/11 (2004)': False,
  'Happy Gilmore (1996)': False,
  'Diamonds are Forever (1971)': False,
  'Scream (1996)': False},
 {'Fahrenheit 9/11 (2004)': False,
  'Happy Gilmore (1996)': False,
  'Diamonds are Forever (1971)': False,
  'Scream (1996)': False})
```

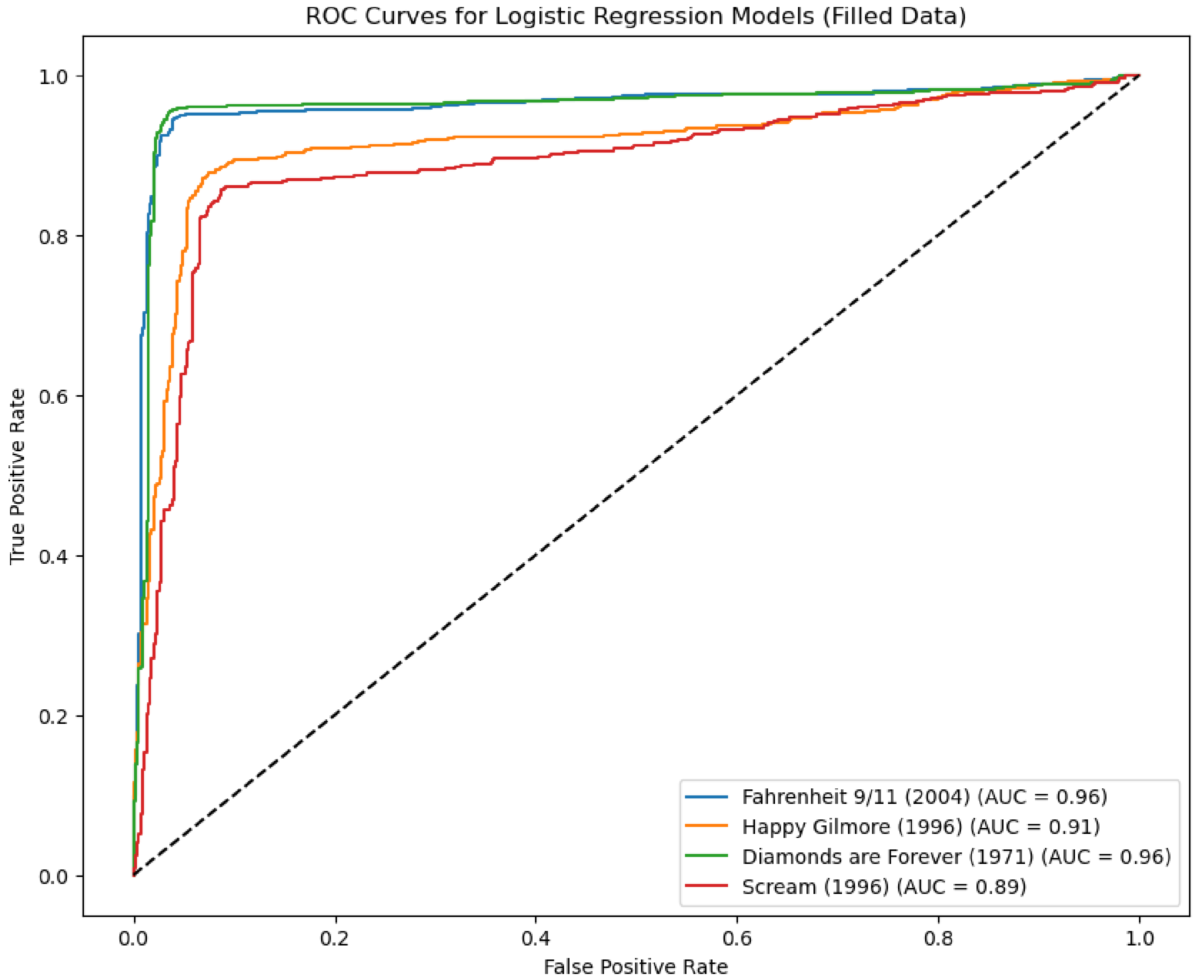
In [40]:
```
# Re-imputing missing values using the 50/50 blend of row mean and colum
movie_ratings_imputed = data.iloc[:, :400].copy()  # Copying the first 4(

# Calculate the mean for each movie and for each user, excluding NaNs
column_means_imputed = movie_ratings_imputed.mean(axis=0, skipna=True)
row_means_imputed = movie_ratings_imputed.mean(axis=1, skipna=True)

# Impute missing values
for row_index, row in movie_ratings_imputed.iterrows():
    for col_index in row[row.isna()].index:
        user_mean = row_means_imputed[row_index]
        movie_mean = column_means_imputed[col_index]
        imputed_value = (user_mean + movie_mean) / 2
        movie_ratings_imputed.at[row_index, col_index] = imputed_value

# Recalculate the user average ratings after imputation
user_average_ratings_imputed = movie_ratings_imputed.mean(axis=1)

# Reshape for compatibility with model input
X_imputed = user_average_ratings_imputed.values.reshape(-1, 1)

# Checking for NaN or infinite values in the re-imputed predictor variab
nan_in_X_imputed = np.isnan(X_imputed).any()
infinite_in_X_imputed = np.isinf(X_imputed).any()

nan_in_X_imputed, infinite_in_X_imputed
```

Out[40]: (True, False)

In [41]:
```
# Calculate the overall average rating of all movies (the average of colu
overall_average_rating = column_means_imputed.mean()

# Assign this overall average rating to users with no ratings
user_average_ratings_imputed_filled = user_average_ratings_imputed.fillna

# Reshape for compatibility with model input
X_imputed_filled = user_average_ratings_imputed_filled.values.reshape(-1

# Re-check for NaN or infinite values in the filled predictor variable (
nan_in_X_imputed_filled = np.isnan(X_imputed_filled).any()
infinite_in_X_imputed_filled = np.isinf(X_imputed_filled).any()

nan_in_X_imputed_filled, infinite_in_X_imputed_filled
```

Out[41]: (False, False)

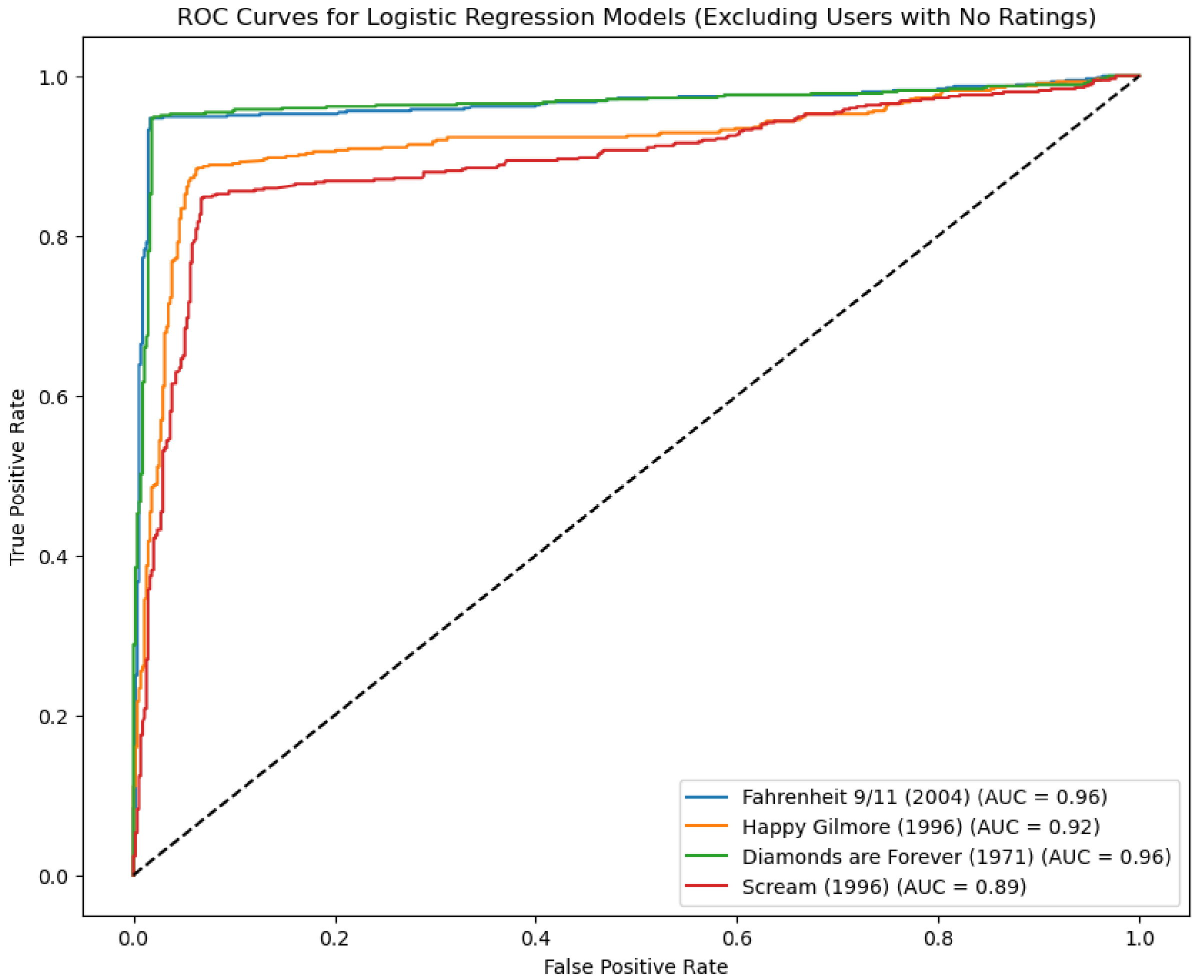
In [42]:
```
# Building logistic regression models for each target movie and performi
model_results_filled = {}

for movie in target_movies_index:
    Y = median_split_df_imputed[movie]

    # Creating the logistic regression model
    log_reg_model_filled = LogisticRegression()

    # Cross-validation for AUC
    cv_auc_scores_filled = cross_val_score(log_reg_model_filled, X_imput

    # Fit the model to the entire dataset for coefficient and ROC analys
    log_reg_model_filled.fit(X_imputed_filled, Y)

    # Coefficients (Betas)
    coefficients_filled = log_reg_model_filled.coef_

    # ROC and AUC
    Y_pred_prob_filled = log_reg_model_filled.predict_proba(X_imputed_fi
    fpr_filled, tpr_filled, _ = roc_curve(Y, Y_pred_prob_filled)
    roc_auc_filled = auc(fpr_filled, tpr_filled)

    # Storing results
    model_results_filled[movie] = {'coefficients': coefficients_filled,
                                   'mean_cv_auc': np.mean(cv_auc_scores_
                                   'roc_auc': roc_auc_filled,
                                   'fpr': fpr_filled, 'tpr': tpr_filled}

# Plotting ROC curves for the models using filled data
plt.figure(figsize=(10, 8))

for movie, result in model_results_filled.items():
    plt.plot(result['fpr'], result['tpr'], label=f'{movie} (AUC = {resul

plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Logistic Regression Models (Filled Data)')
plt.legend(loc='lower right')
plt.show()

# Returning the coefficients and mean cross-validated AUC for each model
filled_model_info = {movie: {'coefficients': result['coefficients'], 'me
                     for movie, result in model_results_filled.items()}
filled_model_info
```



Out[42]:
```
{'Fahrenheit 9/11 (2004)': {'coefficients': array([[9.38409563]]),
  'mean_cv_auc': 0.9619887785275608},
 'Happy Gilmore (1996)': {'coefficients': array([[7.11750404]]),
  'mean_cv_auc': 0.9146999772537721},
 'Diamonds are Forever (1971)': {'coefficients': array([[9.22360618]]),
  'mean_cv_auc': 0.9611985745697172},
 'Scream (1996)': {'coefficients': array([[6.1107945]]),
  'mean_cv_auc': 0.8890008340283571}}
```

In [43]:
```
# Excluding users who have not rated any movies (users with NaN in their
movie_ratings_excluded = data.iloc[:, :400].copy()  # Copying the first 4
user_average_ratings_excluded = movie_ratings_excluded.mean(axis=1)
users_to_exclude = user_average_ratings_excluded.isna()

# Filtering out these users from the dataset
movie_ratings_filtered = movie_ratings_excluded[~users_to_exclude]
user_average_ratings_filtered = user_average_ratings_excluded[~users_to_

# Reshape for compatibility with model input
X_filtered = user_average_ratings_filtered.values.reshape(-1, 1)

# Building logistic regression models for each target movie and performi
model_results_filtered = {}

for movie in target_movies_index:
    Y_filtered = median_split_df_imputed[movie][~users_to_exclude]

    # Creating the logistic regression model
    log_reg_model_filtered = LogisticRegression()

    # Cross-validation for AUC
    cv_auc_scores_filtered = cross_val_score(log_reg_model_filtered, X_f

    # Fit the model to the entire dataset for coefficient and ROC analys
    log_reg_model_filtered.fit(X_filtered, Y_filtered)

    # Coefficients (Betas)
    coefficients_filtered = log_reg_model_filtered.coef_

    # ROC and AUC
    Y_pred_prob_filtered = log_reg_model_filtered.predict_proba(X_filter
    fpr_filtered, tpr_filtered, _ = roc_curve(Y_filtered, Y_pred_prob_fi
    roc_auc_filtered = auc(fpr_filtered, tpr_filtered)

    # Storing results
    model_results_filtered[movie] = {'coefficients': coefficients_filter
                                     'mean_cv_auc': np.mean(cv_auc_score
                                     'roc_auc': roc_auc_filtered,
                                     'fpr': fpr_filtered, 'tpr': tpr_fil

# Plotting ROC curves for the models using filtered data
plt.figure(figsize=(10, 8))

for movie, result in model_results_filtered.items():
    plt.plot(result['fpr'], result['tpr'], label=f'{movie} (AUC = {resul

plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Logistic Regression Models (Excluding Users wi
plt.legend(loc='lower right')
plt.show()

# Returning the coefficients and mean cross-validated AUC for each model
filtered_model_info = {movie: {'coefficients': result['coefficients'], '
                       for movie, result in model_results_filtered.items
```

```
filtered_model_info
```



Out[43]:
```
{'Fahrenheit 9/11 (2004)': {'coefficients': array([[7.39226353]]),
  'mean_cv_auc': 0.9636545606186975},
 'Happy Gilmore (1996)': {'coefficients': array([[5.1991041]]),
  'mean_cv_auc': 0.9171203275456821},
 'Diamonds are Forever (1971)': {'coefficients': array([[7.32150072]]),
  'mean_cv_auc': 0.9648178027143833},
 'Scream (1996)': {'coefficients': array([[4.41378758]]),
  'mean_cv_auc': 0.8921449692925923}}
```

In [44]:
```
#average for missing value in X
```

In [45]:

```
# Replacing NaN values in X with the mean of the non-NaN elements
X_mean = np.nanmean(X)  # Compute the mean of non-NaN elements
X_filled = np.nan_to_num(X, nan=X_mean)  # Replace NaN with the computed

# Confirming that there are no more NaN values in X
nan_in_X_filled = np.isnan(X_filled).any()
infinite_in_X_filled = np.isinf(X_filled).any()

nan_in_X_filled, infinite_in_X_filled
```

Out[45]: (False, False)

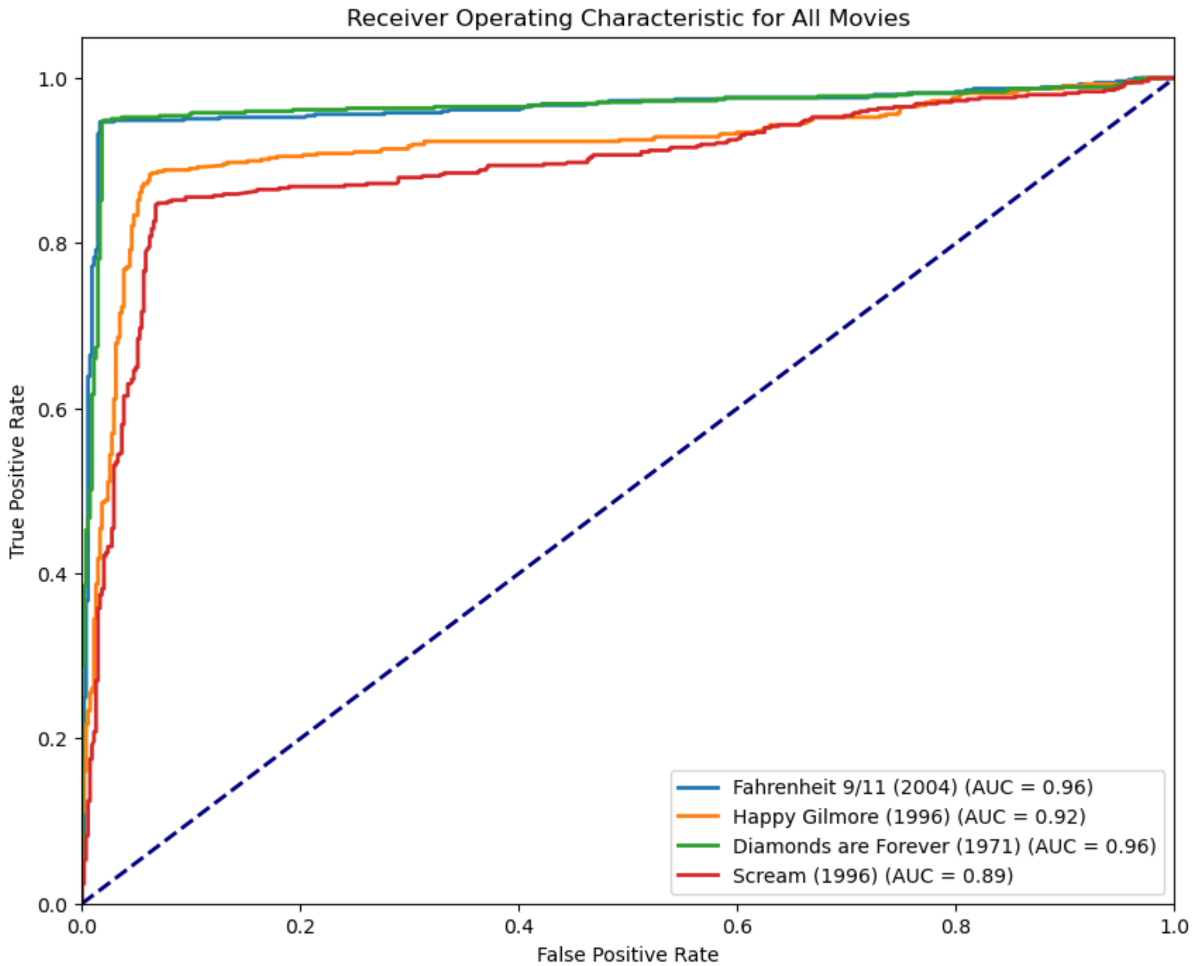
In [46]:

```
# Identifying the columns for the four target movies
target_movies = ["Fahrenheit 9/11 (2004)", "Happy Gilmore (1996)", "Diam

# Creating a DataFrame to hold the binary labels for these movies
target_movie_labels = pd.DataFrame()

# Labeling movie ratings: 1 if rating is above the median (enjoyed), 0 i
for movie in target_movies:
    median_rating = movie_ratings[movie].median()
    labels = movie_ratings[movie].apply(lambda x: 1 if x > median_rating
    target_movie_labels[movie] = labels

target_movie_labels.head()
```

Out[46]:

	Fahrenheit 9/11 (2004)	Happy Gilmore (1996)	Diamonds are Forever (1971)	Scream (1996)
0	0	1	0	0
1	0	0	0	0
2	1	1	1	1
3	0	0	0	0
4	0	0	0	0

In [47]:
```
# Re-building logistic regression models for each movie with the updated
model_results_updated = {}

for movie in target_movies:
    Y = target_movie_labels[movie]

    # Logistic Regression Model
    model = LogisticRegression()

    # Cross-validation (using AUC as the metric)
    auc_scores = cross_val_score(model, X_filled.reshape(-1, 1), Y, cv=5

    # Training the model on the entire dataset for retrieving the beta co
    model.fit(X_filled.reshape(-1, 1), Y)
    beta_coefficient = model.coef_[0][0]

    # Predicting probabilities for ROC curve
    Y_prob = model.predict_proba(X_filled.reshape(-1, 1))[:, 1]
    fpr, tpr, _ = roc_curve(Y, Y_prob)
    roc_auc = auc(fpr, tpr)

    # Storing results
    model_results_updated[movie] = {'AUC Scores': auc_scores, 'Mean AUC'
                                    'Beta Coefficient': beta_coefficient
                                    'FPR': fpr, 'TPR': tpr}

   # Plotting ROC Curves for all four movies in one graph
plt.figure(figsize=(10, 8))

for movie in target_movies:
    fpr = model_results_updated[movie]['FPR']
    tpr = model_results_updated[movie]['TPR']
    roc_auc = model_results_updated[movie]['ROC AUC']
    plt.plot(fpr, tpr, lw=2, label=f'{movie} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for All Movies')
plt.legend(loc="lower right")
plt.show()
```



In [25]:
```
# Extracting the beta coefficients from the previously computed logistic
beta_coefficients = {movie: model_results_updated[movie]['Beta Coefficie
beta_coefficients
```

Out[25]:
```
{'Fahrenheit 9/11 (2004)': 7.396363831892333,
 'Happy Gilmore (1996)': 5.20153199835788,
 'Diamonds are Forever (1971)': 7.325544036473053,
 'Scream (1996)': 4.415679573492213}
```

In [11]:
```
#Perform 80/20 trian/test split on 400 movie ratings based on number of 
#And select movies from several franchises Star Wars,Harry Potter,The Ma
#Toy Story, Batman in this dataset.
#A linear regression model was built for these series and R^2 and MSE we
#To determine if the ratings from a small number of viewers at the first
#used to determine the ratings after the release of the movie.
```

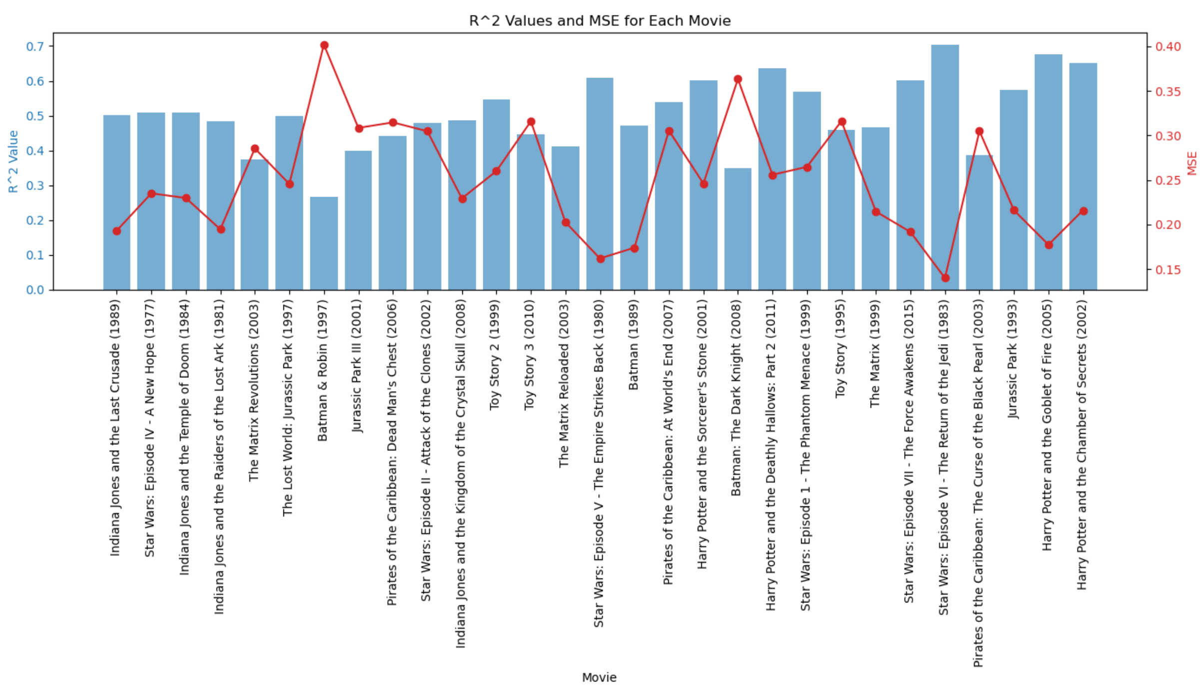
In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Select specific movie franchises
franchises = ['Star Wars', 'Harry Potter', 'The Matrix', 'Indiana Jones'
              'Jurassic Park', 'Pirates of the Caribbean', 'Toy Story',
movie_titles = data.columns[:400]
franchise_movies = [title for title in movie_titles if any(franchise in

# Handle missing data
franchise_data = data[franchise_movies]
column_means = franchise_data.mean(axis=0, skipna=True)
row_means = franchise_data.mean(axis=1, skipna=True)
global_mean = franchise_data.stack().mean()

for row_index, row in franchise_data.iterrows():
    for col_index in row[row.isna()].index:
        user_mean = row_means[row_index] if not pd.isna(row_means[row_in
        movie_mean = column_means[col_index] if not pd.isna(column_means
        imputed_value = (user_mean + movie_mean) / 2
        franchise_data.at[row_index, col_index] = imputed_value

# Split the data into training and test sets
train_data, test_data = train_test_split(franchise_data, test_size=0.2,

# Train the model and calculate R^2 and MSE for each movie
movie_r2_mse = {}
for movie in franchise_movies:
    y_train = train_data[movie]
    y_test = test_data[movie]

    lr_regressor = LinearRegression()
    lr_regressor.fit(train_data.drop(columns=[movie]), y_train)

    predicted_ratings = lr_regressor.predict(test_data.drop(columns=[mov
    r2 = r2_score(y_test, predicted_ratings)
    mse = mean_squared_error(y_test, predicted_ratings)

    movie_r2_mse[movie] = (r2, mse)

# Display the results
for movie, (r2, mse) in movie_r2_mse.items():
    print(f"{movie}: R^2 = {r2:.2f}, MSE = {mse:.2f}")
```

```
Indiana Jones and the Last Crusade (1989): R^2 = 0.50, MSE = 0.19
Star Wars: Episode IV - A New Hope (1977): R^2 = 0.51, MSE = 0.24
Indiana Jones and the Temple of Doom (1984): R^2 = 0.51, MSE = 0.23
Indiana Jones and the Raiders of the Lost Ark (1981): R^2 = 0.48, MSE =
0.19
The Matrix Revolutions (2003): R^2 = 0.37, MSE = 0.29
The Lost World: Jurassic Park (1997): R^2 = 0.50, MSE = 0.25
Batman & Robin (1997): R^2 = 0.27, MSE = 0.40
Jurassic Park III (2001): R^2 = 0.40, MSE = 0.31
Pirates of the Caribbean: Dead Man's Chest (2006): R^2 = 0.44, MSE = 0.
31
Star Wars: Episode II - Attack of the Clones (2002): R^2 = 0.48, MSE =
0.30
Indiana Jones and the Kingdom of the Crystal Skull (2008): R^2 = 0.49,
MSE = 0.23
Toy Story 2 (1999): R^2 = 0.55, MSE = 0.26
Toy Story 3 (2010): R^2 = 0.45, MSE = 0.32
The Matrix Reloaded (2003): R^2 = 0.41, MSE = 0.20
Star Wars: Episode V - The Empire Strikes Back (1980): R^2 = 0.61, MSE
= 0.16
Batman (1989): R^2 = 0.47, MSE = 0.17
Pirates of the Caribbean: At World's End (2007): R^2 = 0.54, MSE = 0.31
Harry Potter and the Sorcerer's Stone (2001): R^2 = 0.60, MSE = 0.25
Batman: The Dark Knight (2008): R^2 = 0.35, MSE = 0.36
Harry Potter and the Deathly Hallows: Part 2 (2011): R^2 = 0.64, MSE =
0.26
Star Wars: Episode 1 - The Phantom Menace (1999): R^2 = 0.57, MSE = 0.2
6
Toy Story (1995): R^2 = 0.46, MSE = 0.32
The Matrix (1999): R^2 = 0.47, MSE = 0.21
Star Wars: Episode VII - The Force Awakens (2015): R^2 = 0.60, MSE = 0.
19
Star Wars: Episode VI - The Return of the Jedi (1983): R^2 = 0.70, MSE
= 0.14
Pirates of the Caribbean: The Curse of the Black Pearl (2003): R^2 = 0.
39, MSE = 0.31
Jurassic Park (1993): R^2 = 0.57, MSE = 0.22
Harry Potter and the Goblet of Fire (2005): R^2 = 0.68, MSE = 0.18
Harry Potter and the Chamber of Secrets (2002): R^2 = 0.65, MSE = 0.22
```

In [13]:

```
# Extracting movie titles and corresponding R^2 and MSE values
movies = list(movie_r2_mse.keys())
r2_values = [movie_r2_mse[movie][0] for movie in movies]
mse_values = [movie_r2_mse[movie][1] for movie in movies]

# Creating subplots
fig, ax1 = plt.subplots(figsize=(14, 8))

# Plotting R^2 values
ax1.set_xlabel('Movie')
ax1.set_ylabel('R^2 Value', color='tab:blue')
ax1.bar(movies, r2_values, color='tab:blue', alpha=0.6, label='R^2 Value
ax1.tick_params(axis='y', labelcolor='tab:blue')
ax1.set_xticklabels(movies, rotation=90)

# Creating a twin axis for MSE values
ax2 = ax1.twinx()
ax2.set_ylabel('MSE', color='tab:red')
ax2.plot(movies, mse_values, color='tab:red', marker='o', label='MSE')
ax2.tick_params(axis='y', labelcolor='tab:red')

# Adding a title and showing the plot
plt.title('R^2 Values and MSE for Each Movie')
fig.tight_layout()
plt.show()
```

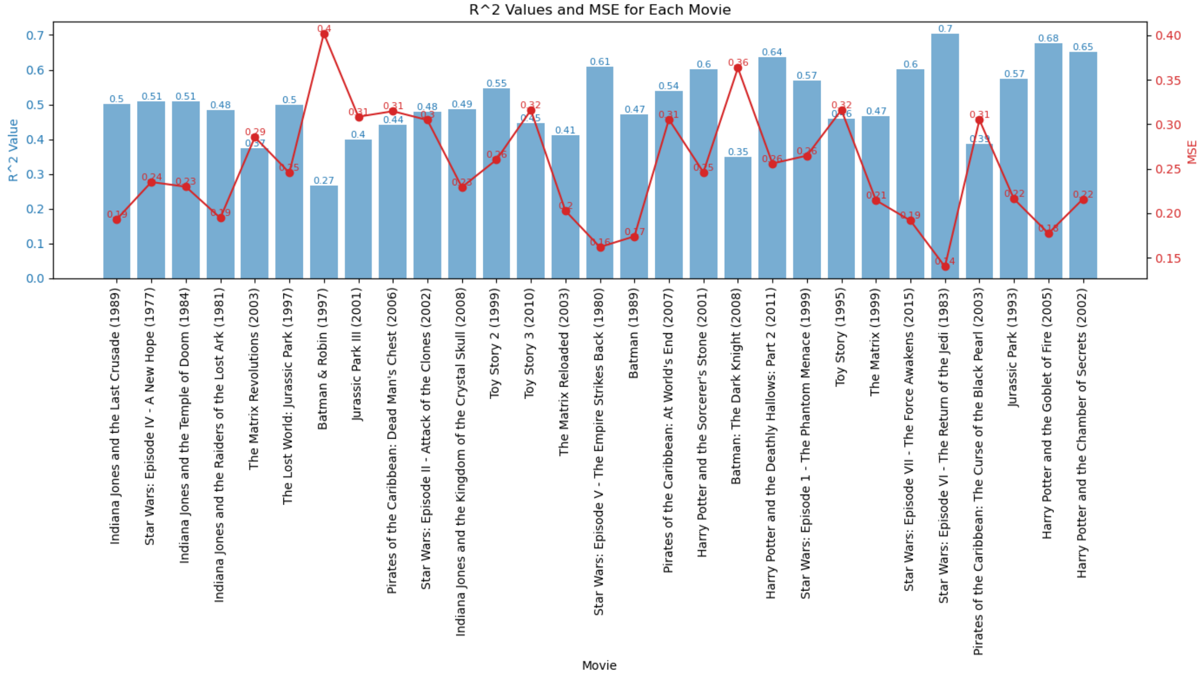
```
/var/folders/hz/65rckc890_v9s165kqqw5_0h0000gn/T/ipykernel_43162/955683
719.py:14: UserWarning: FixedFormatter should only be used together wit
h FixedLocator
  ax1.set_xticklabels(movies, rotation=90)
```



In [14]:

```
# Extracting movie titles and corresponding R^2 and MSE values
movies = list(movie_r2_mse.keys())
r2_values = [movie_r2_mse[movie][0] for movie in movies]
mse_values = [movie_r2_mse[movie][1] for movie in movies]

# Creating subplots
fig, ax1 = plt.subplots(figsize=(14, 8))

# Plotting R^2 values
ax1.set_xlabel('Movie')
ax1.set_ylabel('R^2 Value', color='tab:blue')
bars = ax1.bar(movies, r2_values, color='tab:blue', alpha=0.6, label='R^
ax1.tick_params(axis='y', labelcolor='tab:blue')
ax1.set_xticklabels(movies, rotation=90)

# Adding text annotation for R^2 values
for bar in bars:
    yval = bar.get_height()
    ax1.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), va='

# Creating a twin axis for MSE values
ax2 = ax1.twinx()
ax2.set_ylabel('MSE', color='tab:red')
lines = ax2.plot(movies, mse_values, color='tab:red', marker='o', label=
ax2.tick_params(axis='y', labelcolor='tab:red')

# Adding text annotation for MSE values
for i, line in enumerate(lines[0].get_data()[1]):
    ax2.text(i, line, round(line, 2), va='bottom', ha='center', color='ta

# Adding a title and showing the plot
plt.title('R^2 Values and MSE for Each Movie')
fig.tight_layout()
plt.show()
```

```
/var/folders/hz/65rckc890_v9s165kqqw5_0h0000gn/T/ipykernel_43162/408674
7928.py:14: UserWarning: FixedFormatter should only be used together wi
th FixedLocator
  ax1.set_xticklabels(movies, rotation=90)
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```