

Predicción de películas de IMDB

Integrantes del grupo: Ezequiel Montero, Dana Fernández, De Prada Boris, Gatica German

Introducción

Contexto del Problema:

El desafío consiste en desarrollar un algoritmo que permita predecir la categoría de películas a mostrar, considerando variables como género, país de producción y valoraciones de directores y actores.

Importancia y Relevancia:

La importancia de establecer reglas para entrenar nuestro modelo, es indicar cuales son sus variables más importantes, de esta manera de acuerdo a la información que le proponemos nuestro modelo entrenará de una manera o de otra.

Objetivos del Proyecto:

Utilizando datos de la base de datos de IMDB, que incluyen información sobre títulos, años de producción, actores y directores, este proyecto busca analizar cómo estos datos contribuyen a mejorar las predicciones. El objetivo final es desarrollar un modelo capaz de prever las puntuaciones de calificación de IMDB, considerando una gama amplia de atributos, como duración, elenco y títulos. Este enfoque integral busca ofrecer una evaluación precisa del potencial rendimiento de una película en la plataforma de IMDB.

Metodología

Los datos que empleamos están almacenados en un repositorio de GitHub que incluye varios conjuntos de datos. Hemos pasado por diversas etapas de limpieza y fortalecimiento de modelos predictivos con el fin de llevar a cabo la clasificación de películas según la valoración proporcionada por IMDB. Es crucial determinar las variables esenciales que nuestro modelo requiere para realizar predicciones precisas y aproximarse al objetivo deseado.

Herramientas y Tecnologías

Estas herramientas y tecnologías, combinadas con el lenguaje de programación Python, nos permitieron llevar a cabo un análisis exhaustivo de los datos y construir modelos de machine learning eficientes y precisos.

Manipulación y análisis de datos: Utilizamos Numpy y Pandas para manipular y analizar eficientemente conjuntos de datos, aprovechando las estructuras de datos y funciones que ofrecen.

Visualización de datos: Seaborn y Matplotlib fueron las herramientas elegidas para visualizar datos de manera efectiva. Seaborn proporciona estilos estéticos y funciones estadísticas, mientras que Matplotlib es una librería de visualización más general.

Construir modelos de machine learning: Empleamos Sklearn, una librería versátil que abarca diversas técnicas de machine learning, desde regresión lineal hasta clasificación y agrupación. Además, incorporamos XG Boost, una implementación eficiente de árboles de decisión y boosting.

Matriz de Correlación: Utilizamos la Matriz de Correlación para entender las relaciones entre las variables, lo que es esencial para comprender la estructura de los datos y su impacto en la predicción.

Random Forest: Implementamos Random Forest como un método robusto para regresión y clasificación, aprovechando la fortaleza de múltiples árboles de decisión.

Árbol de Decisión: Los Árboles de Decisión fueron empleados para modelar decisiones basadas en reglas, lo que proporciona una comprensión interpretativa de la toma de decisiones del modelo.

Clasificador KNeighbors: Implementamos el clasificador KNeighbors para realizar clasificación basada en la proximidad de los vecinos más cercanos, una técnica útil para problemas de clasificación.

XG Boost: Utilizamos XG Boost como una potente herramienta de boosting que mejora la precisión de los modelos, especialmente útil en conjuntos de datos complejos.

Proceso de Análisis/Desarrollo:

- 1) Como primer paso, realizamos la limpieza de nuestros datos, buscamos datos repetidos o que contengan NaN, etc.

```
imdb_data = imdb_data.dropna() # elimina toda la fila donde
                               encuentre un NaN

# Verificar duplicados
duplicates = imdb_data[imdb_data.duplicated()]

# Contar el número total de filas duplicadas
num_duplicates = len(duplicates)

# Eliminar filas duplicadas
movie_data_no_duplicates = imdb_data.drop_duplicates()
```

- 2) Eliminamos aquellas columnas que no tengan datos numéricos.

```
#Elimino las variables que no son numericas
movies =
movies.drop(['director_name','actor_2_name','genres','language','country','color',
'movie_title','content_rating','movie_imdb_link','actor_1_name','plot_keywords','a
ctor_3_name'],axis = 1)
```

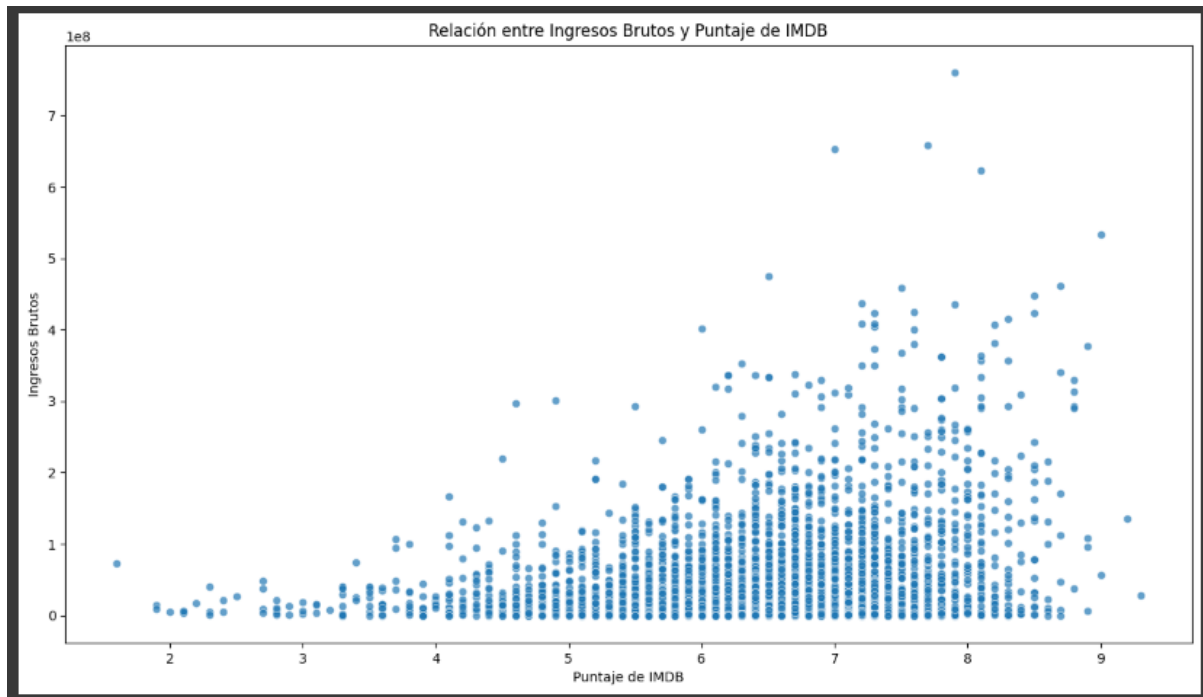
- 3) Creamos un Dataframe con los datos nuevos.

```
movies = movie_data_no_duplicates
pd.set_option('display.max_columns', None) #para que me muestre todas las
columnas
movies.head(10)
```

● Presentación de Resultados:

- 1) Creamos el gráfico de dispersión:

```
# Crear el gráfico de dispersión
plt.figure(figsize=(15, 8))
sns.scatterplot(x='imdb_score', y='gross', data=movie_data_no_duplicates,
alpha=0.7)
plt.title('Relación entre Ingresos Brutos y Puntaje de IMDB')
plt.xlabel('Ingresos Brutos')
plt.ylabel('Puntaje de IMDB')
plt.show()
```



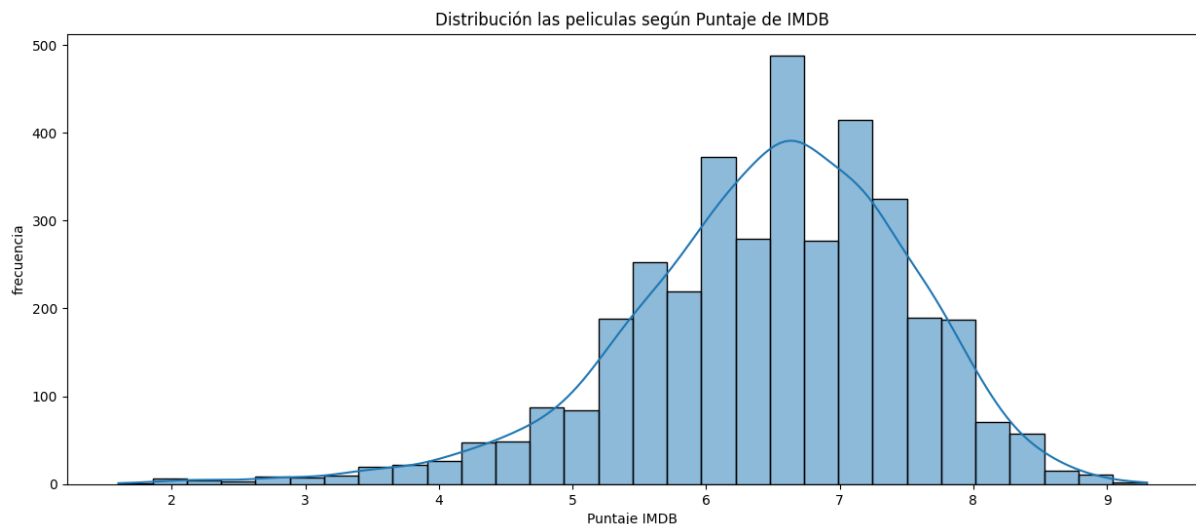
El gráfico de dispersión visualiza la relación entre el puntaje de IMDb y los ingresos brutos de las películas en el conjunto de datos. Cada punto en el gráfico representa una película, siendo su posición determinada por su puntaje de IMDb en el eje horizontal (X) y sus ingresos brutos en el eje vertical (Y).

Al observar el gráfico, podemos notar una tendencia general: a medida que el puntaje de IMDb aumenta, los ingresos brutos también tienden a aumentar. Esta observación sugiere una correlación positiva entre la calidad percibida de una película, medida por el puntaje de IMDb, y su éxito en taquilla, medido por los ingresos brutos. En otras palabras, las películas con puntajes de IMDb más altos tienden a tener mayores ingresos brutos.

2) Creamos un histograma con los datos:

```
# Visualizar la distribución --- HISTOGRAMA
plt.figure(figsize=(15, 6))
sns.histplot(movie_data_no_duplicates['imdb_score'], bins=30, kde=True)
plt.title('Distribución las peliculas según Puntaje de IMDB')
plt.xlabel('Puntaje IMDB')
plt.ylabel('frecuencia')
plt.show()
```

Output:



El histograma generado representa la distribución de los puntajes de IMDb en el conjunto de datos de películas.

En el eje X (Horizontal): Representa los diferentes valores de puntajes de IMDb. En este caso, se divide en 30 intervalos (bins) para capturar la variabilidad en los puntajes.

Eje Y (Vertical): Indica la frecuencia o la cantidad de películas que tienen un determinado puntaje de IMDb. Cuanto mayor sea la barra en un intervalo particular, más películas tienen un puntaje dentro de ese rango.

La forma del histograma muestra cómo se distribuyen los puntajes en el conjunto de datos. Podemos observar que la distribución es simétrica.

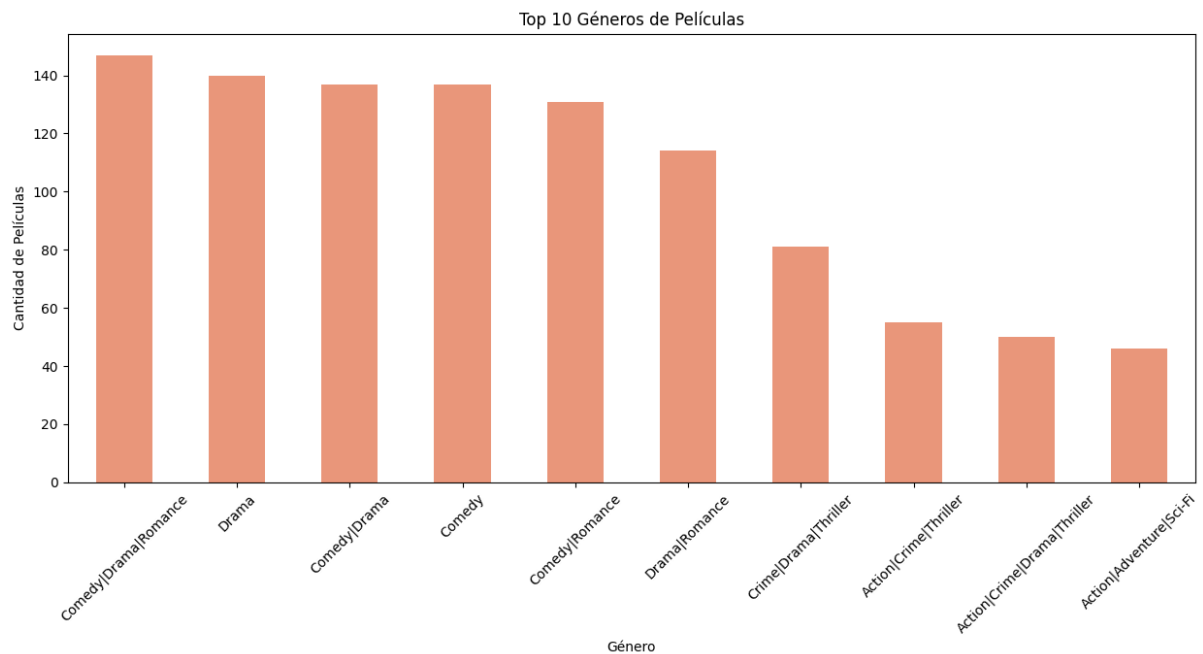
La curva suavizada sobre el histograma es una estimación de la densidad de probabilidad de los puntajes. Proporciona una visualización suavizada de la distribución, ayudando a identificar patrones generales.

1) Mostramos gráfico de barra:

```
genero_count = movies['genres'].value_counts().head(10) # Top 10 géneros más comunes

# Crear el gráfico de barras
plt.figure(figsize=(15, 6))
genero_count.plot(kind='bar', color= '#E9967A')
plt.title('Top 10 Géneros de Películas')
plt.xlabel('Género')
plt.ylabel('Cantidad de Películas')
plt.xticks(rotation=45) # Rotar las etiquetas en el eje x para una mejor legibilidad
plt.show()
```

Su resultado:



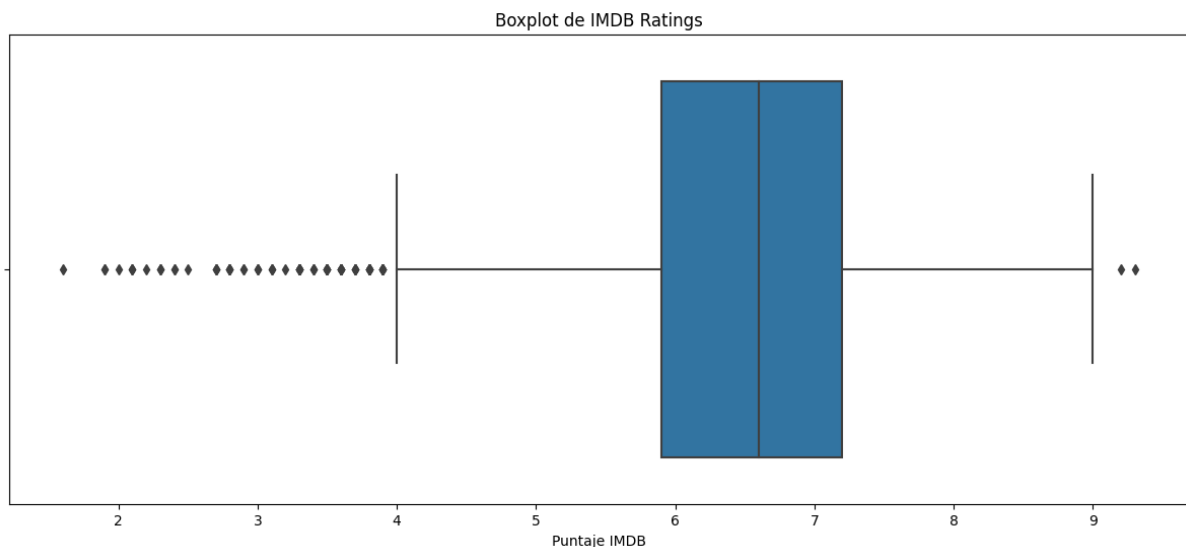
El gráfico de barras representa los diez géneros de películas más comunes en el conjunto de datos. Cada barra en el gráfico corresponde a un género específico, y la altura de la barra indica la cantidad de películas asociadas con ese género.

Al observar el gráfico, puedes identificar rápidamente los géneros predominantes en la muestra analizada.

1) Realizamos un Boxplot:

```
# Boxplot para identificar outliers en el puntaje de IMDB
plt.figure(figsize=(15, 6))
sns.boxplot(x=movie_data_no_duplicates['imdb_score'])
plt.title('Boxplot de IMDB Ratings')
plt.xlabel('Puntaje IMDB')
plt.show()
```

Resultado:



Caja (Box): Representa el rango intercuartil (IQR), que abarca desde el primer cuartil (Q1) hasta el tercer cuartil (Q3). La línea en el medio de la caja es la mediana (Q2). La caja en sí encapsula el 50% central de los datos.

Bigotes (Whiskers): Representan la extensión de los datos hasta cierto límite. Los puntos fuera de los bigotes se consideran outliers.

Puntos fuera de los Bigotes: Indican posibles outliers, valores atípicos que se encuentran significativamente más allá del rango intercuartil.

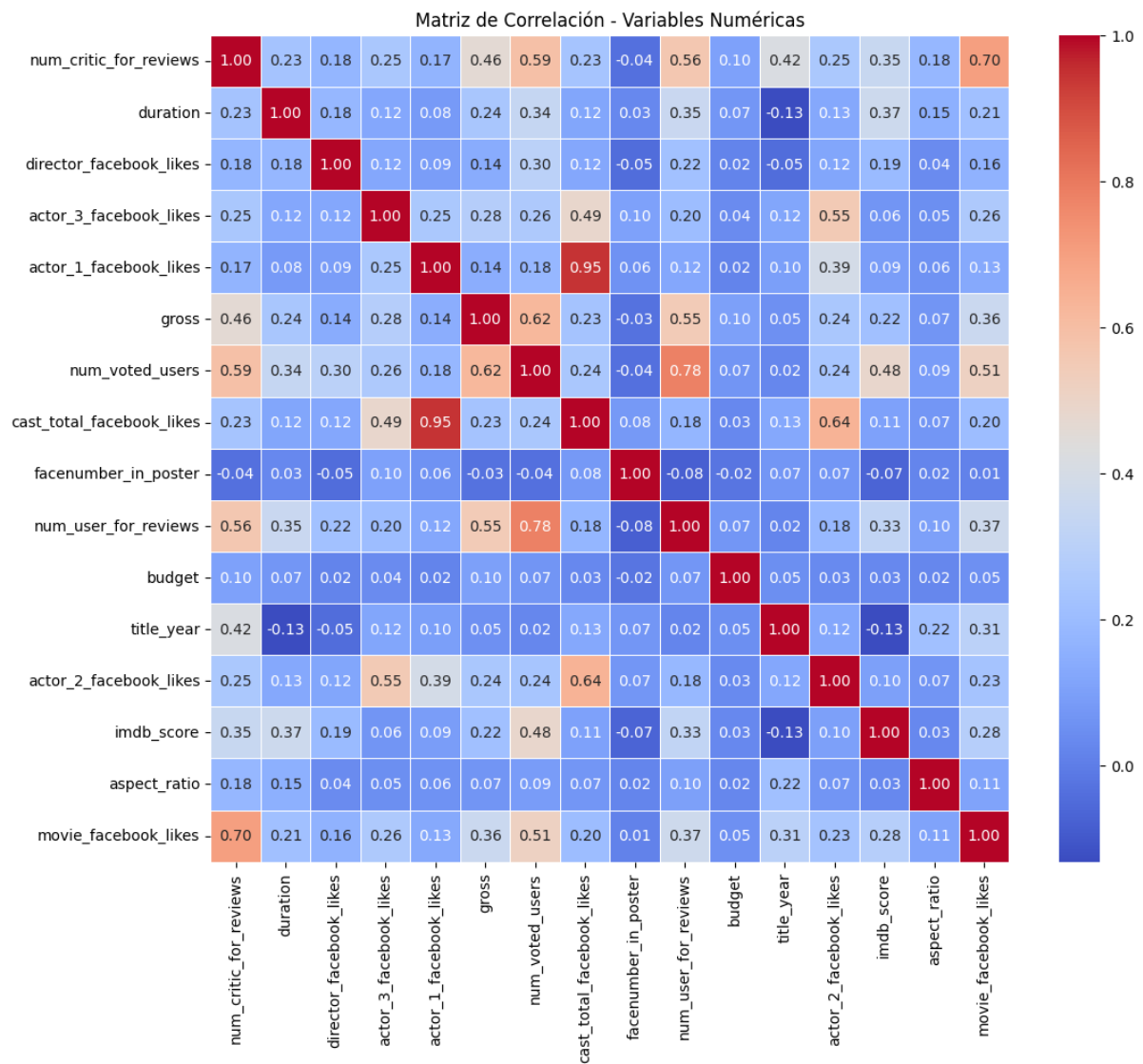
En esta instancia, se observa que el 50% de los datos se encuentra comprendido entre los valores 5,80 y 7,20. A continuación, los valores que se extienden desde 5,80 hasta el cuartil inferior, que es 4 (límite inferior), y desde el cuartil superior, 7,30, hasta 9, constituyen los "Whiskers" o extremos del diagrama de caja. Los valores que se encuentran por debajo de 4 y por encima de 9 se clasifican como outliers, denotando observaciones atípicas o excepcionales en el contexto de la distribución de los puntajes de IMDb en el conjunto de datos.

2) Generamos la Matriz de correlación:

```
matriz_correlacion = movies.corr()

# Visualizar la matriz de correlación
plt.figure(figsize=(12, 10))
sns.heatmap(matriz_correlacion, annot=True, cmap='coolwarm', fmt='.2f',
            linewidths=0.5)
plt.title('Matriz de Correlación - Variables Numéricas')
plt.show()
```

Salida del código:



Al examinar el mapa de calor de la matriz de correlación, podemos realizar observaciones inmediatas sobre las relaciones entre las variables. En particular, nuestra variable dependiente, `imdb_score`, muestra una correlación más significativa con ciertas variables numéricas. Las variables que destacan en términos de correlación con **`imdb_score`** son **`num_user_for_review`**, **`num_voted_users`**, **`duration`**, y **`num_critic_for_view`**.

- **Matriz de Correlación:** Es una tabla que muestra las correlaciones entre todas las variables numéricas en el conjunto de datos. Cada celda de la matriz contiene un valor que indica la fuerza y la dirección de la relación lineal entre dos variables. Los valores están en el rango de -1 a 1.
 - **Colores en el Mapa de Calor:** La intensidad del color en el mapa de calor representa la magnitud de la correlación. Colores más intensos (ya sea en azul o rojo) indican una correlación más fuerte, mientras que colores más tenues indican una correlación más débil.
 - **Valores Anotados:** Cada celda de la matriz tiene un número anotado, que representa el coeficiente de correlación. Este número puede ser negativo (indicando una correlación negativa), positivo (indicando una correlación positiva), o cercano a cero (indicando una correlación débil o nula).
 - **Rango de Valores:** Los valores en la matriz de correlación están en el rango de -1 a 1. Un valor de -1 indica una correlación negativa perfecta, 1 indica una correlación positiva perfecta y 0 indica que no hay correlación lineal.
- Interpretación: Explicar qué significan estos resultados en el contexto del problema.

```
# Seleccionamos las características más correlacionadas con imdb_score
target_correlation =
matriz_correlacion['imdb_score'].abs().sort_values(ascending=False)

# Mostramos las características más correlacionadas
selected_features_correlation = target_correlation[1:6] # Seleccionar las 5
características más importantes
selected_features_correlation

Resultado:
num_voted_users      0.482583
duration             0.367388
num_critic_for_reviews 0.349825
num_user_for_reviews 0.325026
movie_facebook_likes 0.284034
```

Codificación One-Hot para variables categóricas o nominales

Convertimos una variable categórica en una representación binaria, donde cada categoría se representa como un vector de bits, y solo un bit es "encendido" (establecido en 1) para indicar la pertenencia a esa categoría. Convertir una variable categórica en una representación binaria, donde cada categoría se representa como un vector de bits, y solo un bit es "encendido" (establecido en 1) para indicar la pertenencia a esa categoría.

También, utilizamos una funcionalidad del modelo Random Forest que analiza y busca las características más importantes de los datos (`rf_model.feature_importances_`) para complementar con el trabajo que hizo el método con la matriz de correlación y verificar si hay coincidencias en cuanto a la selección de las variables más importantes, solo que en este caso, le pediremos que nos muestre no solo 5 sino las 15 características más importantes, que son, (luego de obtener resultados positivos ya que ambos análisis nos muestran como más relevantes a las mismas características), las que utilizaremos posteriormente para trabajar con los modelos de aprendizaje automático que hemos elegido.

```
movie_data_encoded = pd.get_dummies(movies)

#Separar la variable objetivo 'imdb_score' de las características
X = movie_data_encoded.drop('imdb_score', axis=1)
y = movie_data_encoded['imdb_score']

#Inicializar un modelo de Random Forest
rf_model = RandomForestRegressor(random_state=42)

#Ajustar el modelo y obtener la importancia de las características
rf_model.fit(X, y)
feature_importance = rf_model.feature_importances_

#Crear un DataFrame con la importancia de las características
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance':
feature_importance})

#Seleccionar las características más importantes (por ejemplo, las 5 más
importantes)
top_features = feature_importance_df.sort_values(by='Importance',
ascending=False).head(15)['Feature'].tolist()

#Mostrar las características más importantes
print(top_features)
```

Vamos a definir los límites de nuestro bucket. Estos límites dividen el rango de valores posibles de la variable original en intervalos específicos. En el segundo paso, se asigna a cada bucket una etiqueta. En este caso, las etiquetas son "**Bad**", "**OK**", "**Good**" y "**Excellent**".

```

# Definir los límites para los buckets . Estos límites dividen el rango de
valores posibles de la variable original en intervalos específicos.
bins = [0, 4, 6, 8, 10]

#Aignar a cada bucket una etiqueta
labels = ['Bad', 'OK', 'Good', 'Excellent']

# Crear la variable objetivo agrupada en buckets. y_grouped contendrá las
etiquetas asociadas a cada valor de y según el bucket al que pertenezca
y_grouped = pd.cut(y, bins=bins, labels=labels)

# Mostrar las primeras filas de la variable objetivo agrupada
print(y_grouped.head())

Resultado:
0      Good
1      Good
2      Good
3  Excellent
5      Good
Name: imdb_score, dtype: category
Categories (4, object): ['Bad' < 'OK' < 'Good' < 'Excellent']

```

El modelo que utilizamos a continuación es un bosque aleatorio, un tipo de algoritmo de aprendizaje automático supervisado que se utiliza para la clasificación y la regresión. En la primera línea, dividimos el conjunto de datos en conjuntos de entrenamiento y prueba. El conjunto de entrenamiento se utilizará para entrenar el modelo, y el conjunto de prueba se utilizará para evaluar el rendimiento del modelo.

Como resultado de la evaluación, se muestran un porcentaje en forma de error cuadrático medio (MSE) y coeficiente de determinación (R^2). El MSE es una medida de la precisión de las predicciones del modelo, y el R^2 es una medida de la relación lineal entre las predicciones del modelo y los valores reales.

```

#divido el conjunto de datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Seleccionar solo las características más importantes
X_train_selected = X_train[top_features]
X_test_selected = X_test[top_features]

# Inicializar un modelo de bosque aleatorio con las características
seleccionadas
model_selected = RandomForestRegressor(random_state=42)

# Entrenar el modelo en el conjunto de entrenamiento con características
seleccionadas
model_selected.fit(X_train_selected, y_train)

# Realizar predicciones en el conjunto de prueba con características
seleccionadas
y_pred_selected = model_selected.predict(X_test_selected)

# Evaluar el rendimiento del modelo con características seleccionadas
mse_selected = mean_squared_error(y_test, y_pred_selected)
r2_selected = r2_score(y_test, y_pred_selected)

print(f'Mean Squared Error (con características seleccionadas): {mse_selected}')
print(f'R^2 Score (con características seleccionadas): {r2_selected}')

```

Resultado:

```

Mean Squared Error (con características seleccionadas): 0.44689698523489935
R^2 Score (con características seleccionadas): 0.563774879863497

```

Decision Tree Classifier

En este caso vamos a seleccionar las 15 variables o características que previamente obtuvimos como las más relevantes para el análisis. Dividimos los datos en conjunto de entrenamiento y de prueba, utilizando el 20% de los datos al conjunto de prueba.

```
#Supongamos que 'selected_features' es la lista de características relevantes
seleccionadas previamente
selected_features = ['num_voted_users', 'duration', 'budget', 'gross',
'num_user_for_reviews', 'num_critic_for_reviews', 'title_year',
'actor_3_facebook_likes', 'director_facebook_likes',
'actor_2_facebook_likes', 'cast_total_facebook_likes',
'movie_facebook_likes', 'actor_1_facebook_likes', 'content_rating_PG-13',
'facenumber_in_poster']

# Seleccionar solo las características relevantes y la variable objetivo
agrupada
X_selected_grouped = X[selected_features]
X_train_selected_grouped, X_test_selected_grouped, y_train_grouped,
y_test_grouped = train_test_split(X_selected_grouped, y_grouped, test_size=0.2,
random_state=42)

# Inicializar el modelo de árbol de clasificación
dt_classifier_selected_grouped = DecisionTreeClassifier(random_state=42)

# Entrenar el modelo en el conjunto de entrenamiento
dt_classifier_selected_grouped.fit(X_train_selected_grouped, y_train_grouped)

# Realizar predicciones en el conjunto de prueba
y_pred_selected_grouped =
dt_classifier_selected_grouped.predict(X_test_selected_grouped)

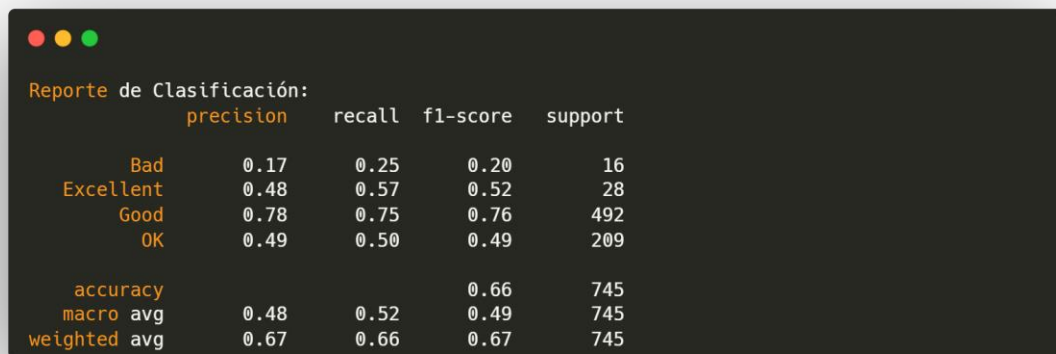
# Calcular la precisión del modelo
accuracy_selected_grouped = accuracy_score(y_test_grouped,
y_pred_selected_grouped)
print(f'Accuracy: {accuracy_selected_grouped}')
```

Este es su resultado, teniendo como objetivo predecir la variable objetivo agrupada en función de las características seleccionadas:

```
Accuracy: 0.6604026845637584
```

Graficamos la matriz de confusión, es una tabla que muestra la distribución de las predicciones del modelo en comparación con los valores reales. La utilizamos para evaluar el rendimiento del modelo en términos de precisión, sensibilidad y especificidad.

El reporte de clasificación:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays a classification report for a model. The report includes a table with columns for precision, recall, f1-score, and support, grouped by class (Bad, Excellent, Good, OK) and then by overall metrics (accuracy, macro avg, weighted avg).

Reporte de Clasificación:				
	precision	recall	f1-score	support
Bad	0.17	0.25	0.20	16
Excellent	0.48	0.57	0.52	28
Good	0.78	0.75	0.76	492
OK	0.49	0.50	0.49	209
accuracy			0.66	745
macro avg	0.48	0.52	0.49	745
weighted avg	0.67	0.66	0.67	745

El modelo tiene una precisión del 66%, lo que significa que realizó correctamente el 66% de las predicciones, tiene una sensibilidad del 52%, lo que significa que identificó correctamente el 52% de las películas de cada categoría. El modelo tiene una especificidad del 52%, lo que significa que identificó correctamente el 52% de las películas que no pertenecían a cada categoría.

K Neighbors Classifier

En este modelo de machine learning volvemos a utilizar los mismos datos que con el modelo anterior, dejando el 20% de los datos para utilizarlos para prueba.

```

# Inicializar el modelo KNN
knn_classifier = KNeighborsClassifier(n_neighbors=5) # Puedes ajustar el número
de vecinos según tus preferencias

# Entrenar el modelo en el conjunto de entrenamiento
knn_classifier.fit(X_train_selected_grouped, y_train_grouped)

# Realizar predicciones en el conjunto de prueba
y_pred_knn = knn_classifier.predict(X_test_selected_grouped)

# Calcular la precisión del modelo KNN
accuracy_knn = accuracy_score(y_test_grouped, y_pred_knn)
print(f'Accuracy KNN: {accuracy_knn}')

# Obtener la matriz de confusión para KNN
conf_matrix_knn = confusion_matrix(y_test_grouped, y_pred_knn)

# Mostrar la matriz de confusión para KNN
print("\nMatriz de Confusión para KNN:")
print(conf_matrix_knn)

# Obtener el reporte de clasificación para KNN
class_report_knn = classification_report(y_test_grouped, y_pred_knn)

# Mostrar el reporte de clasificación para KNN
print("\nReporte de Clasificación para KNN:")
print(class_report_knn)

```

Como resultado, obtenemos los siguientes resultados:

```

Accuracy KNN: 0.6322147651006711

Reporte de Clasificación para KNN:
      precision    recall  f1-score   support

    Bad           0.00      0.00      0.00        16
  Excellent      0.29      0.07      0.11        28
      Good       0.68      0.85      0.76       492
       OK       0.42      0.25      0.31       209

 accuracy          0.63          0.63          0.63       745
  macro avg          0.35          0.29          0.30       745
weighted avg          0.58          0.63          0.59       745

```

El modelo contiene un 63% (correctamente clasificó el 63% de las instancias en el conjunto de prueba). Una precisión promedio por clase del 35% y una sensibilidad promedio por clase: 29% (promedio de las sensibilidades individuales de cada clase).

Clasificador XGBoost

XGBoost (Extreme Gradient Boosting) es un algoritmo de aprendizaje supervisado que se utiliza tanto para problemas de regresión como de clasificación. Es una implementación eficiente y altamente efectiva del algoritmo de aumento de gradiente.

Como primer paso debemos instalarlo:

```
pip install xgboost
```

Con este modelo, vamos a utilizar 14 variables o características que consideramos relevantes para su análisis, como cantidad de votos de usuarios, duración, presupuesto, presupuesto, etc. Dividimos el conjunto de datos en conjuntos de entrenamiento y prueba (80% para entrenamiento y 20% para prueba).

El resultado es:

```
Reporte de Clasificación para XGBoost:
      precision    recall  f1-score   support

    Bad          0.00      0.00      0.00        16
  Excellent      0.76      0.57      0.65        28
    Good         0.83      0.91      0.87       492
    OK           0.71      0.61      0.65       209

 accuracy          0.79        745
  macro avg          0.57        745
 weighted avg          0.77        745
```

- Precisión general: 79% (correctamente clasificó el 79% de las instancias en el conjunto de prueba).
- Precisión promedio por clase: 57% (promedio de las precisiones individuales de cada clase).
- Sensibilidad promedio por clase: 52% (promedio de las sensibilidades individuales de cada clase).

Regresión Lineal

Volvemos a utilizar el mismo método que es utilizar el 80% para entrenamiento y el 20% para pruebas.

```
#Inicializar un modelo de regresión lineal
linear_model = LinearRegression()

# Entrenar el modelo en el conjunto de entrenamiento con todas las
características
linear_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba con todas las características
y_pred_linear = linear_model.predict(X_test)

# Evaluar el rendimiento del modelo de regresión lineal con todas las
características
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print(f'Mean Squared Error (regresión lineal con todas las características):
{mse_linear}')
print(f'R^2 Score (regresión lineal con todas las características):
{r2_linear}')
```

```
# Obtener los coeficientes y la ordenada al origen
m = linear_model.coef_
c = linear_model.intercept_

print(f'Coeficientes asociados a cada variable independiente (normalizados):
{m}')
```

```
print(f'Ordenada al origen (normalizada): {c}')
```

Como resultado del modelo:

```
Mean Squared Error (regresión lineal con todas las características): 0.59475669559003
R^2 Score (regresión lineal con todas las características): 0.4194460478416997
Coeficientes asociados a cada variable independiente (normalizados): [3.15695539e-03
8.15464572e-03 1.83253979e-05 ... 2.15625822e-01
3.51656375e-02 1.34000011e-01]
Ordenada al origen (normalizada): 71.37379020395466
```

En este caso, el MSE es de 0.59. Esto significa que, en promedio, las predicciones del modelo están a 0.59 de los valores reales. el R^2 es de 0.42. Esto significa que el modelo explica el 42% de la variabilidad de la variable dependiente. El MSE es relativamente bajo, lo que

significa que las predicciones del modelo están relativamente cerca de los valores reales. El R^2 también es relativamente alto, lo que significa que el modelo explica una parte significativa de la variabilidad de la variable dependiente.

Random Forest Regression

En este caso del modelo, es similar al anterior con una diferencia, utilizamos el parámetro **random_state** para establecer una semilla aleatoria. El código en este caso entrena el modelo en el conjunto de entrenamiento completo. Esto puede mejorar el rendimiento del modelo, pero también puede aumentar el riesgo de sobreajuste.

```
#Inicializar un modelo de bosque aleatorio para regresión
rf_regressor = RandomForestRegressor(random_state=42)

#Entrenar el modelo en el conjunto de entrenamiento completo
rf_regressor.fit(X_train, y_train)

#Realizar predicciones en el conjunto de prueba
y_pred_rf = rf_regressor.predict(X_test)

#Evaluar el rendimiento del modelo de bosque aleatorio para regresión
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f'Mean Squared Error (Random Forest para regresión): {mse_rf}')
print(f'R^2 Score (Random Forest para regresión): {r2_rf}')
```

Como resultado:

```
Mean Squared Error (Random Forest para regresión): 0.4335573597315436
R^2 Score (Random Forest para regresión): 0.5767959561518472
```

En este caso, las predicciones del modelo están a 0.43 de los valores reales. El R^2 es de 0.58. Esto significa que el modelo explica el 58% de la variabilidad de la variable dependiente. Estos resultados indican que el modelo de regresión forestal aleatorio tiene un rendimiento aceptable. El MSE es relativamente bajo, lo que significa que las predicciones del modelo están relativamente cerca de los valores reales. El R^2 también es relativamente alto, lo que significa que el modelo explica una parte significativa de la variabilidad de la variable dependiente.

A través de este resultado podemos obtener las 3 características:

```

feature_importances = rf_regressor.feature_importances_
feature_importances

//Obtener importancia de características
feature_importances = rf_regressor.feature_importances_

//Crear una lista de tuplas (característica, importancia)
feature_importance_tuples = list(zip(X_train.columns, feature_importances))

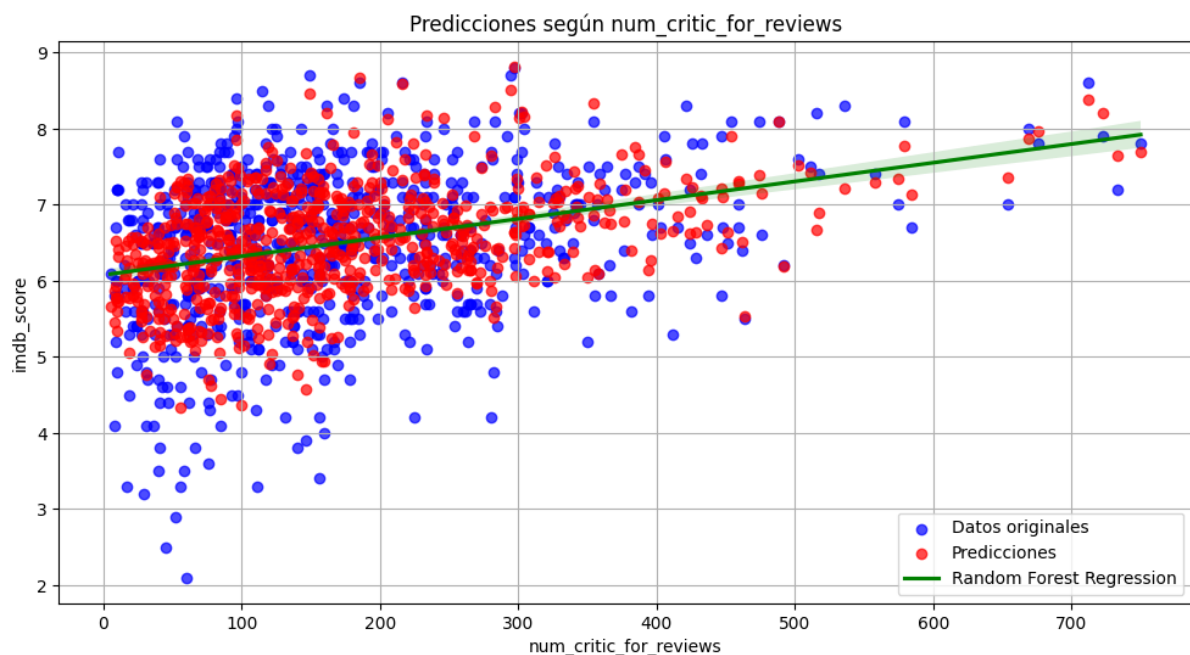
//Ordenar la lista por importancia en orden descendente
feature_importance_tuples_sorted = sorted(feature_importance_tuples, key=lambda x: x[1],
reverse=True)

top_features = feature_importance_tuples_sorted[:3]

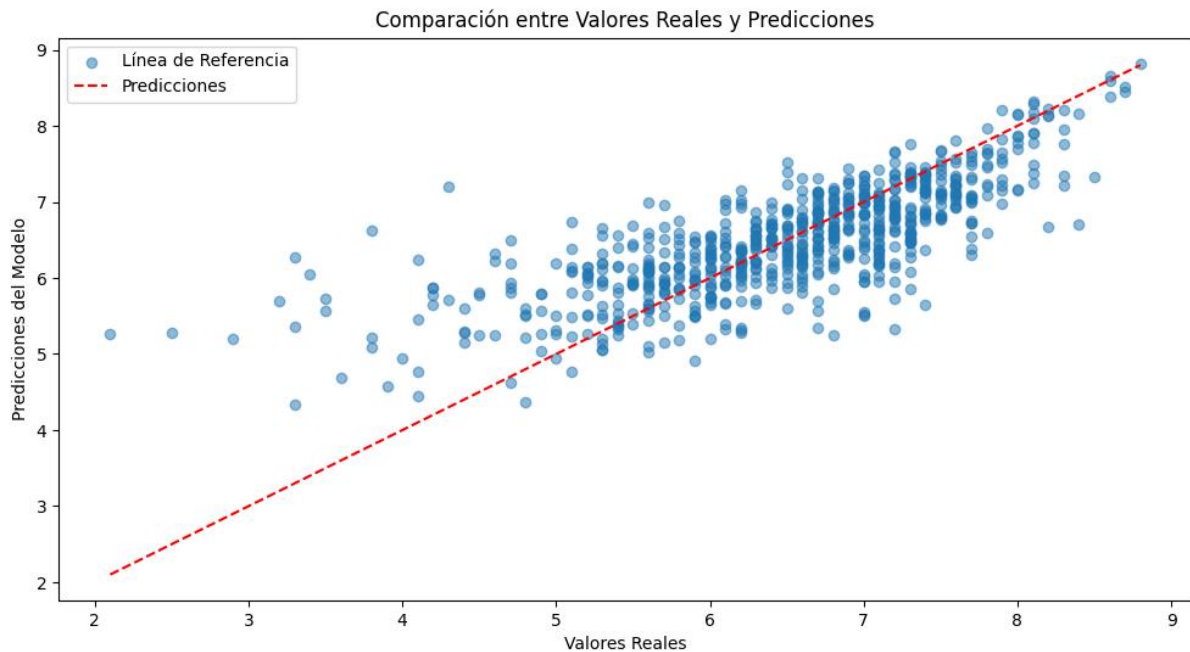
print("Las 3 características más importantes:")
for feature, importance in top_features:
    print(f"{feature}: {importance}")

```

Con esta información realizamos el gráfico, donde graficamos los puntos originales y las predicciones según la variable **num_critic_for_reviews**, que es la que influye en el modelo.



A partir de esto graficamos los valores reales frente a las predicciones del modelo de regresión forestal aleatorio con todas las características.



Conclusiones

En general, los diferentes modelos que entrenamos tienen un rendimiento aceptable. El modelo de regresión forestal aleatorio tiene un rendimiento ligeramente mejor que el modelo de regresión lineal. Esto se debe a que el modelo de regresión forestal aleatorio puede aprender relaciones complejas entre las variables, mientras que el modelo de regresión lineal sólo puede aprender relaciones lineales.

En este caso en particular elegimos un problema que podríamos resolverlo aplicando regresión, ya que las etiquetas originales previstas tienen un valor continuo, representando la puntuación de IMDB, por eso para poder aplicarlos a otros modelos de clasificación y generar otras alternativas al problema planteado tuvimos que generar etiquetas en formato categórico como 'Bad', 'OK', 'Good' y 'Excellent'. En este sentido las películas que contienen el rango entre 0 y 4 son 'Bad', entre 4 y 6 son 'OK', entre 6 y 8 son 'Good' y por encima de 8 son 'Excellent'.

La conversión de las puntuaciones continuas en categorías como 'Bad', 'OK', 'Good', 'Excellent' es una forma de discretizar y simplificar el problema. A menudo, se realiza utilizando criterios como rangos de valores. Por ejemplo, se podría decidir que las películas con puntuaciones entre 0 y 4 son 'Bad', entre 4 y 6 son 'OK', entre 6 y 8 son 'Good', y por encima de 8 son 'Excellent'.

De esta manera, aquello que teníamos como problema de regresión, es decir, predecir una puntuación continua ampliamos su campo en un problema de clasificación, es decir, predecir

una categoría discreta. Por este motivo utilizamos algoritmos como Árbol de Decisión y KNN que están más adaptados para trabajar con este tipo de problemas.

- Sugerencias para Futuras Investigaciones:

Como observación para próximas investigamos se podrían intentar eliminar valores atípicos de los datos, ampliando las características de los mismos. Como consecuencia de esta ampliación podemos utilizar una validación cruzada para seleccionar los parámetros del modelo, intentando reducir al máximo el MSE bajo el concepto de que el modelo está haciendo buenas predicciones. Un MSE alto nos indicaría que el modelo está haciendo malas predicciones.

7. Entrega de Código y Documentación

- Enlace a Google Colab: Proporcionar el enlace al Google Colab donde se realizó el trabajo.
- Documentación del Código: Asegurarse de que el código esté bien documentado y comentado.