```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

```
1 df = pd.read_csv("/content/netflix-rotten-tomatoes-metacritic-imdb.csv")
```

```
1 df.shape
```

```
(9425, 29)
```

```
1 # Split the categorical variables and keep only the first string
2 categorical_columns = ["Genre", "Tags", "Languages", "Director", "Writer", "Actors","Country Availability"]
3 for col in categorical_columns:
4     df[col] = df[col].str.split(',').str[0]
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9425 entries, 0 to 9424
Data columns (total 29 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Title                  9425 non-null   object
 1   Genre                  9400 non-null   object
 2   Tags                   9389 non-null   object
 3   Languages              9266 non-null   object
 4   Series or Movie        9425 non-null   object
 5   Hidden Gem Score       9415 non-null   float64
 6   Country Availability   9414 non-null   object
 7   Runtime                9424 non-null   object
 8   Director               7120 non-null   object
 9   Writer                 7615 non-null   object
 10  Actors                 9314 non-null   object
 11  View Rating            6827 non-null   object
 12  IMDb Score             9417 non-null   float64
 13  Rotten Tomatoes Score  5445 non-null   float64
 14  Metacritic Score       4082 non-null   float64
 15  Awards Received        5226 non-null   float64
 16  Awards Nominated For   6376 non-null   float64
 17  Boxoffice              3754 non-null   float64
 18  Release Date           9217 non-null   datetime64[ns]
 19  Netflix Release Date   9425 non-null   datetime64[ns]
 20  Production House       4393 non-null   object
 21  Netflix Link           9425 non-null   object
 22  IMDb Link              9101 non-null   object
 23  Summary                9420 non-null   object
 24  IMDb Votes             9415 non-null   float64
 25  Image                  9425 non-null   object
 26  Poster                 8487 non-null   object
 27  TMDb Trailer           9425 non-null   object
 28  Trailer Site           9424 non-null   object
dtypes: datetime64[ns](2), float64(8), object(19)
memory usage: 2.1+ MB
```

```
1 df.head()
```

	Title	Genre	Tags	Languages	Series or Movie	Hidden Gem Score	Country Availability	Runtime	Director	Writer	...	Netflix Release Date	Productio Hous
0	Lets Fight Ghost	Crime	Comedy Programmes	Swedish	Series	4.3	Thailand	< 30 minutes	Tomas Alfredson	John Ajvide Lindqvist	...	2021-03-04	Canal· Sandre Metronom
1	HOW TO BUILD A GIRL	Comedy	Dramas	English	Movie	7.0	Canada	1-2 hour	Coky Giedroyc	Caitlin Moran	...	2021-03-04	Film · Monument Picture Lionsga
2	The Con-Heartist	Comedy	Romantic Comedies	Thai	Movie	8.6	Thailand	> 2 hrs	Mez Tharatorn	Pattaranad Bhiboonsawade	...	2021-03-03	Na
3	Gleboka woda	Drama	TV Dramas	Polish	Series	8.7	Poland	< 30 minutes	NaN	NaN	...	2021-03-03	Na
4	Only a Mother	Drama	Social Issue Dramas	Swedish	Movie	8.3	Lithuania	1-2 hour	Alf Sjöberg	Ivar Lo-Johansson	...	2021-03-03	Na

5 rows × 29 columns

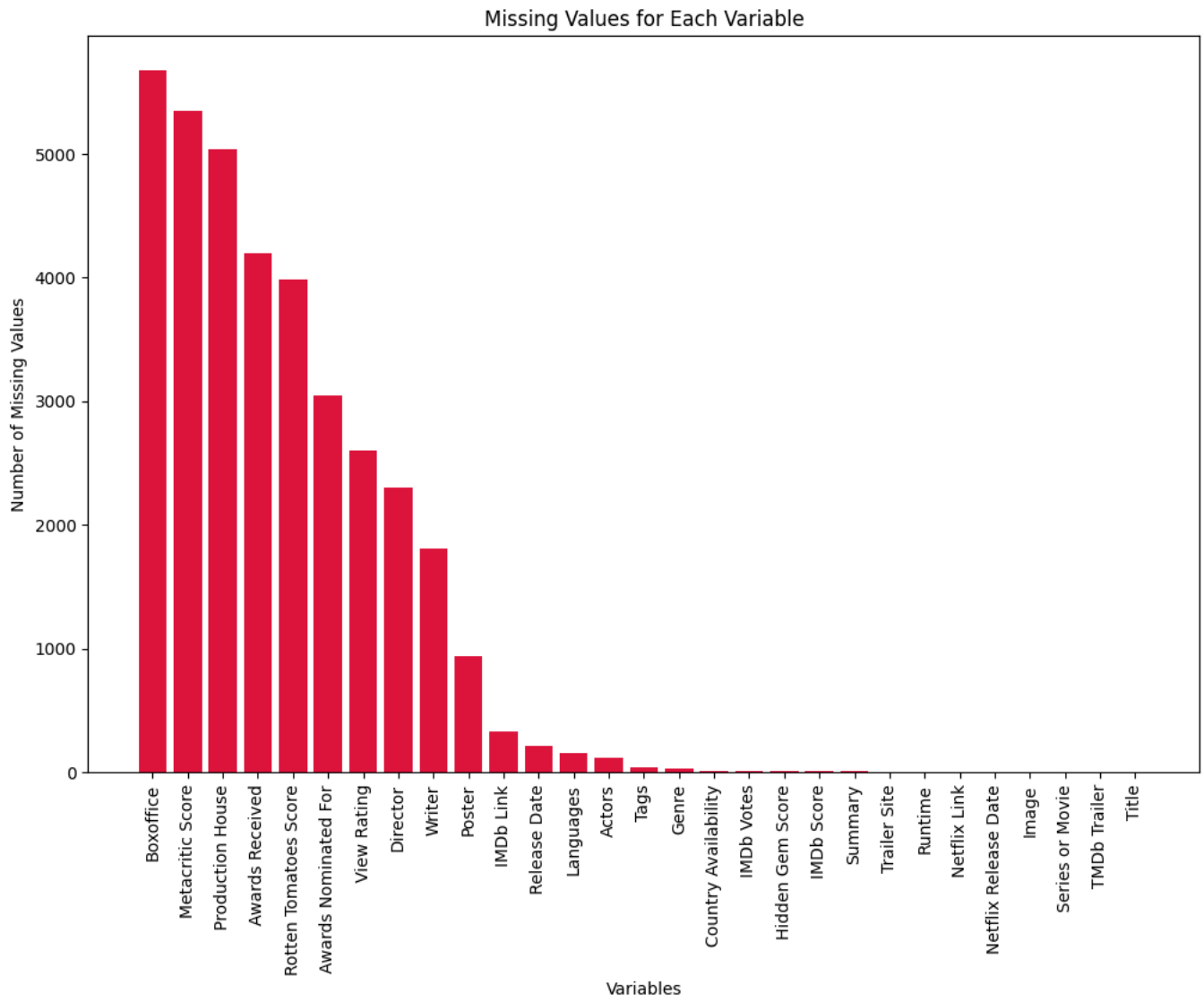
```
1 df.isna().sum()
```

```
Title                     0
Genre                    25
Tags                     36
Languages               159
Series or Movie           0
Hidden Gem Score         10
Country Availability     11
Runtime                   1
Director               2305
Writer                 1810
Actors                  111
View Rating            2598
IMDb Score                8
Rotten Tomatoes Score  3980
Metacritic Score       5343
Awards Received        4199
Awards Nominated For   3049
Boxoffice              5671
Release Date            208
Netflix Release Date      0
Production House       5032
Netflix Link              0
IMDb Link               324
Summary                   5
IMDb Votes               10
Image                     0
Poster                  938
TMDb Trailer              0
Trailer Site              1
dtype: int64
```

Missing Values

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3
4 # Define the variables and their corresponding missing value counts
5 variables = ['Title', 'Genre', 'Tags', 'Languages', 'Series or Movie', 'Hidden Gem Score',
6              'Country Availability', 'Runtime', 'Director', 'Writer', 'Actors', 'View Rating',
7              'IMDb Score', 'Rotten Tomatoes Score', 'Metacritic Score', 'Awards Received',
```

```
            'Awards Nominated For', 'Boxoffice', 'Release Date', 'Netflix Release Date',
            'Production House', 'Netflix Link', 'IMDb Link', 'Summary', 'IMDb Votes',
            'Image', 'Poster', 'TMDb Trailer', 'Trailer Site']

missing_values = [0, 25, 36, 159, 0, 10, 11, 1, 2305, 1810, 111, 2598, 8, 3980, 5343, 4199, 3049,
                  5671, 208, 0, 5032, 0, 324, 5, 10, 0, 938, 0, 1]

# Create a DataFrame to store the variables and their missing value counts
missing_data_df = pd.DataFrame({'Variable': variables, 'Missing Values': missing_values})

# Sort the DataFrame by missing value counts in descending order
missing_data_df.sort_values(by='Missing Values', ascending=False, inplace=True)

# Create a bar plot to visualize missing values
plt.figure(figsize=(12, 8))
plt.bar(missing_data_df['Variable'], missing_data_df['Missing Values'], color='crimson')
plt.xticks(rotation=90)
plt.xlabel('Variables')
plt.ylabel('Number of Missing Values')
plt.title('Missing Values for Each Variable')
plt.show()

```



Columns dropped with more than 50% values missing

```
threshold = len(df) * 0.5  # Calculate the threshold for 50% missing values
subset_df = df.dropna(thresh=threshold, axis=1)

# Print the columns dropped
```

```
dropped_columns = set(df.columns) - set(subset_df.columns)

print(dropped_columns)

df = subset_df
```

```
{'Boxoffice', 'Metacritic Score', 'Production House'}
```

Impute values

```
# Calculate the 5% threshold for null values
threshold = len(df) * 0.05

# List of numerical columns (int or float)
numerical_columns = ['Hidden Gem Score', 'IMDb Score', 'Rotten Tomatoes Score',  'Awards Received', 'Awards Nominated For',

# List of categorical columns (object type)
categorical_columns = ['Genre', 'Tags', 'Languages', 'Series or Movie', 'Country Availability', 'Runtime', 'Director', 'Write

# Replace null values in numerical columns if the count is less than 5% of the DataFrame
for column in numerical_columns:
    if df[column].isnull().sum() < threshold:
        df[column].fillna(df[column].mean(), inplace=True)

# Replace null values in categorical columns if the count is less than 5% of the DataFrame
for column in categorical_columns:
    if df[column].isnull().sum() < threshold:
        df[column].fillna(df[column].mode().iloc[0], inplace=True)



```

```
<ipython-input-10-78c8223369d0>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df[column].fillna(df[column].mean(), inplace=True)
```

```
df['Director'].fillna('Unknown', inplace=True)
df['Writer'].fillna('Unknown', inplace=True)
```

```
df.drop(columns=['View Rating', 'Poster','TMDb Trailer','Image','Trailer Site','Netflix Link','IMDb Link','Summary'], inplace
```

```
df['Awards Received'].fillna(0, inplace=True)
df['Awards Nominated For'].fillna(0, inplace=True)
```

```
# Calculate the mean of 'Rotten Tomatoes Score'
rotten_tomatoes_mean = df['Rotten Tomatoes Score'].mean()

# Replace null values in 'Rotten Tomatoes Score' column with its mean
df['Rotten Tomatoes Score'].fillna(rotten_tomatoes_mean, inplace=True)
```

```
df = df.dropna(subset=['Release Date'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Title                 9217 non-null   object
 1   Genre                 9217 non-null   object
 2   Tags                  9217 non-null   object
 3   Languages             9217 non-null   object
 4   Series or Movie       9217 non-null   object
 5   Hidden Gem Score      9217 non-null   float64
 6   Country Availability  9217 non-null   object
 7   Runtime               9217 non-null   object
 8   Director              9217 non-null   object
 9   Writer                9217 non-null   object
 10  Actors                9217 non-null   object
```

```
 11  IMDb Score             9217 non-null   float64
 12  Rotten Tomatoes Score  9217 non-null   float64
 13  Awards Received        9217 non-null   float64
 14  Awards Nominated For   9217 non-null   float64
 15  Release Date           9217 non-null   datetime64[ns]
 16  Netflix Release Date   9217 non-null   datetime64[ns]
 17  IMDb Votes             9217 non-null   float64
dtypes: datetime64[ns](2), float64(6), object(10)
memory usage: 1.3+ MB
```

```
df['Release Date'] = pd.to_datetime(df['Release Date'])

# Find the last date in the 'Release Date' column
last_date = df['Release Date'].max()

# Calculate the number of days from 'Release Date' till the last date
df['Days_Since_Movie_Released'] = (last_date - df['Release Date']).dt.days
```

```
<ipython-input-17-8fb3f2ad1857>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Release Date'] = pd.to_datetime(df['Release Date'])
<ipython-input-17-8fb3f2ad1857>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Days_Since_Movie_Released'] = (last_date - df['Release Date']).dt.days
```

Extracting year and month for further analysis of release date

```
# Extracting year and month from 'Release Date'
df['Release Year'] = df['Release Date'].dt.year
df['Release Month'] = df['Release Date'].dt.month
```

```
<ipython-input-18-36b4f31718c4>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Release Year'] = df['Release Date'].dt.year
<ipython-input-18-36b4f31718c4>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Release Month'] = df['Release Date'].dt.month
```

```
df['Netflix Release Date'] = pd.to_datetime(df['Netflix Release Date'])

# Find the last date in the 'Release Date' column
last_date = df['Netflix Release Date'].max()

# Calculate the number of days from 'Release Date' till the last date
df['Days_Since_Movie_Released_on_Netflix'] = (last_date - df['Netflix Release Date']).dt.days
```

```
<ipython-input-19-caa7da3ce69c>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Netflix Release Date'] = pd.to_datetime(df['Netflix Release Date'])
<ipython-input-19-caa7da3ce69c>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Days_Since_Movie_Released_on_Netflix'] = (last_date - df['Netflix Release Date']).dt.days
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 22 columns):
 #   Column                                Non-Null Count  Dtype
```

```
 ---  ------                                 --------------  -----
  0   Title                                  9217 non-null   object
  1   Genre                                  9217 non-null   object
  2   Tags                                   9217 non-null   object
  3   Languages                              9217 non-null   object
  4   Series or Movie                        9217 non-null   object
  5   Hidden Gem Score                       9217 non-null   float64
  6   Country Availability                   9217 non-null   object
  7   Runtime                                9217 non-null   object
  8   Director                               9217 non-null   object
  9   Writer                                 9217 non-null   object
  10  Actors                                 9217 non-null   object
  11  IMDb Score                             9217 non-null   float64
  12  Rotten Tomatoes Score                  9217 non-null   float64
  13  Awards Received                        9217 non-null   float64
  14  Awards Nominated For                   9217 non-null   float64
  15  Release Date                           9217 non-null   datetime64[ns]
  16  Netflix Release Date                   9217 non-null   datetime64[ns]
  17  IMDb Votes                             9217 non-null   float64
  18  Days_Since_Movie_Released              9217 non-null   int64
  19  Release Year                           9217 non-null   int64
  20  Release Month                          9217 non-null   int64
  21  Days_Since_Movie_Released_on_Netflix   9217 non-null   int64
dtypes: datetime64[ns](2), float64(6), int64(4), object(10)
memory usage: 1.6+ MB
```

```
df.drop(['Release Date','Netflix Release Date'],axis=1,inplace=True)
```

```
<ipython-input-21-1f093ac1e93f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df.drop(['Release Date','Netflix Release Date'],axis=1,inplace=True)
```

```
df.info()
```

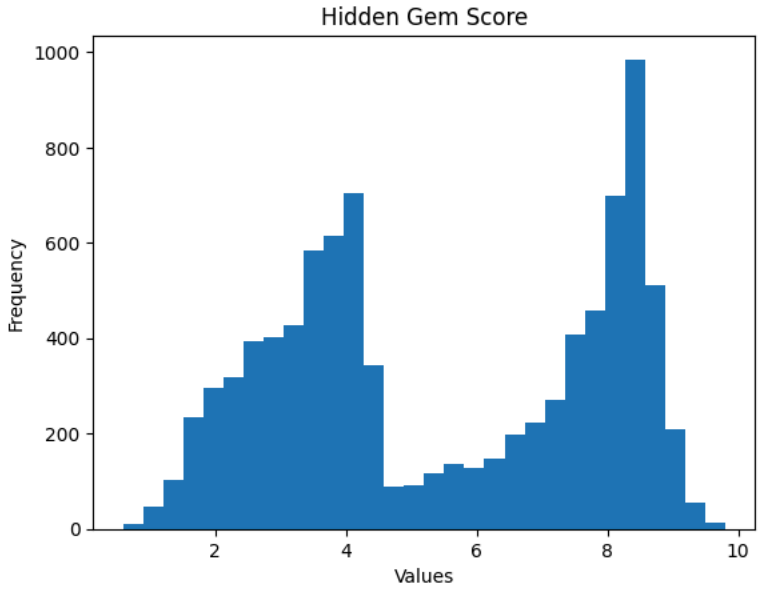
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 20 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   Title                                  9217 non-null   object
 1   Genre                                  9217 non-null   object
 2   Tags                                   9217 non-null   object
 3   Languages                              9217 non-null   object
 4   Series or Movie                        9217 non-null   object
 5   Hidden Gem Score                       9217 non-null   float64
 6   Country Availability                   9217 non-null   object
 7   Runtime                                9217 non-null   object
 8   Director                               9217 non-null   object
 9   Writer                                 9217 non-null   object
 10  Actors                                 9217 non-null   object
 11  IMDb Score                             9217 non-null   float64
 12  Rotten Tomatoes Score                  9217 non-null   float64
 13  Awards Received                        9217 non-null   float64
 14  Awards Nominated For                   9217 non-null   float64
 15  IMDb Votes                             9217 non-null   float64
 16  Days_Since_Movie_Released              9217 non-null   int64
 17  Release Year                           9217 non-null   int64
 18  Release Month                          9217 non-null   int64
 19  Days_Since_Movie_Released_on_Netflix   9217 non-null   int64
dtypes: float64(6), int64(4), object(10)
memory usage: 1.5+ MB
```

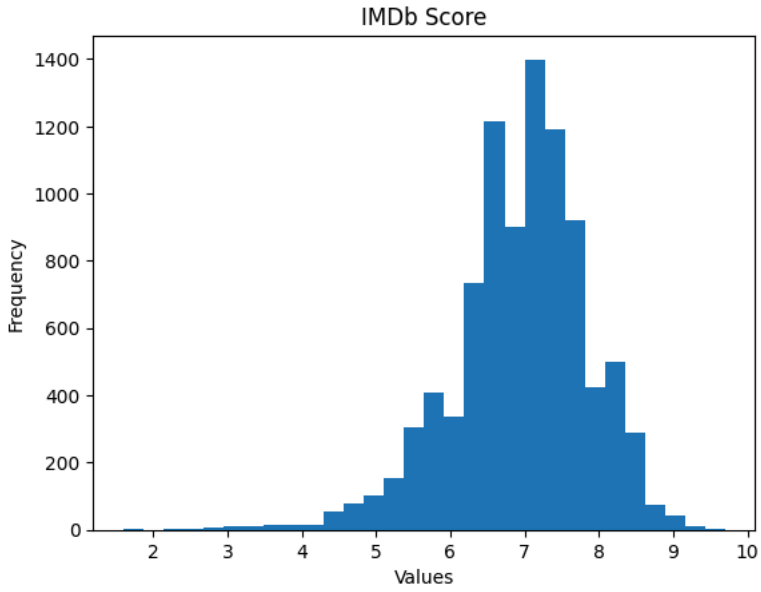
Checking Outliers

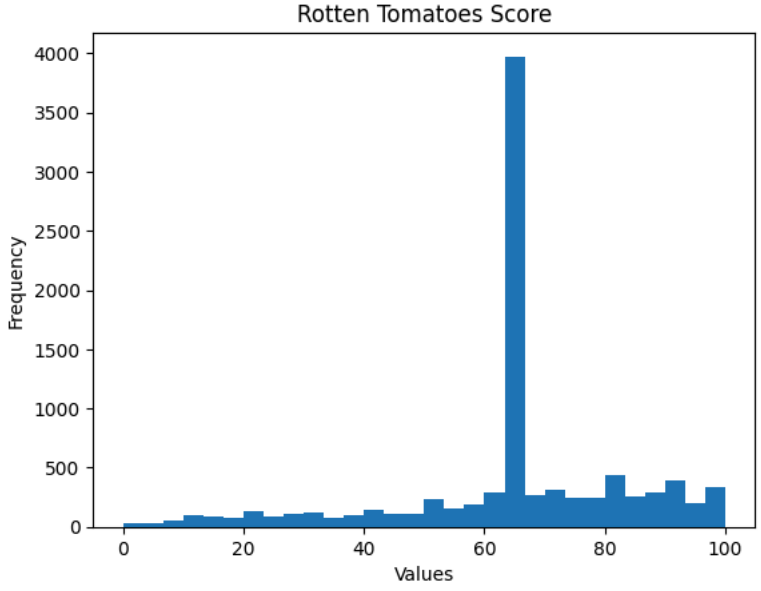
```
numerical_columns = ["Hidden Gem Score", "IMDb Score", "Rotten Tomatoes Score",
                     "Awards Received", "Awards Nominated For", "IMDb Votes","Days_Since_Movie_Released","Days_Since_Movie_Rele


```

```
import matplotlib.pyplot as plt

# Assuming your data is stored in a DataFrame called 'df' and 'column' represents the column of interest
dict_skew = {}
for i in numerical_columns:
  plt.hist(df[i], bins=30)
```

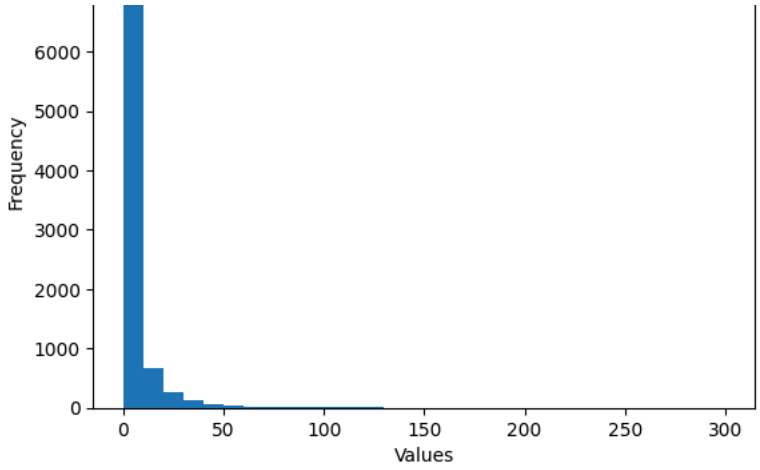
```
  plt.xlabel('Values')
  plt.ylabel('Frequency')
  plt.title(i)
  plt.show()
  data = df[i]
  skewness = np.mean((data - np.mean(data))**3) / np.power(np.var(data), 1.5)
  dict_skew[i] = skewness

for i,k in dict_skew.items():
  print(f"Pearson's First Coefficient of Skewness for {i}:", k)


```

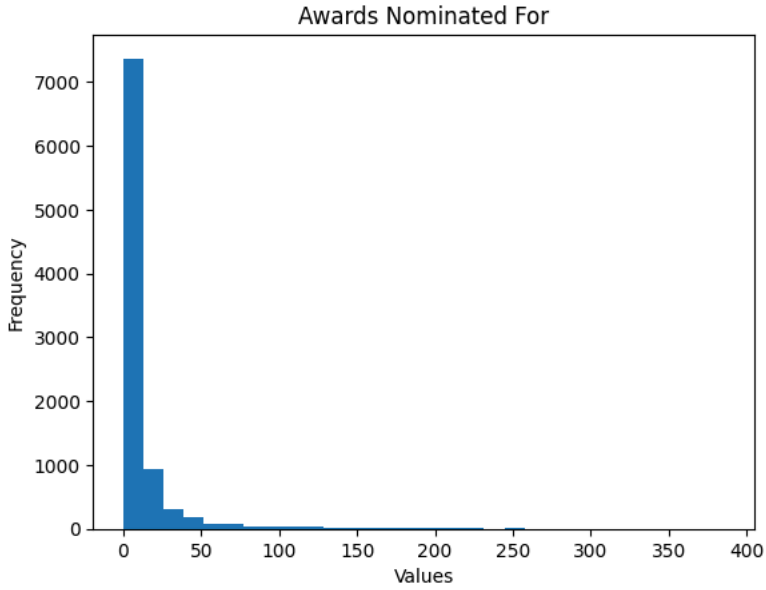

Hidden Gem Score
Frequency
Values
0
200
400
600
800
1000
2
4
6
8
10

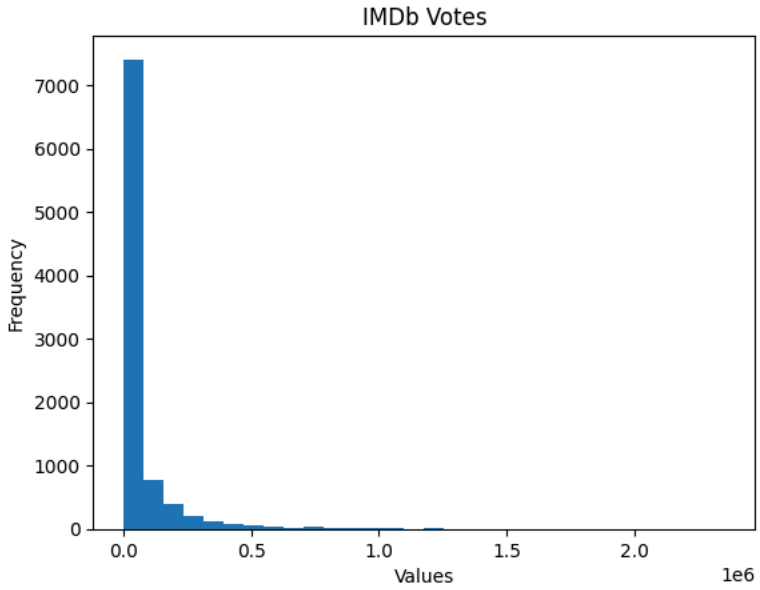

IMDb Score
Frequency
Values
0
200
400
600
800
1000
1200
1400
2
3
4
5
6
7
8
9
10

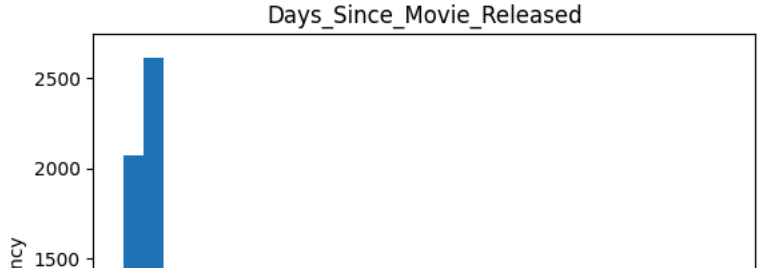

Rotten Tomatoes Score
Frequency
Values
0
500
1000
1500
2000
2500
3000
3500
4000
0
20
40
60
80
100

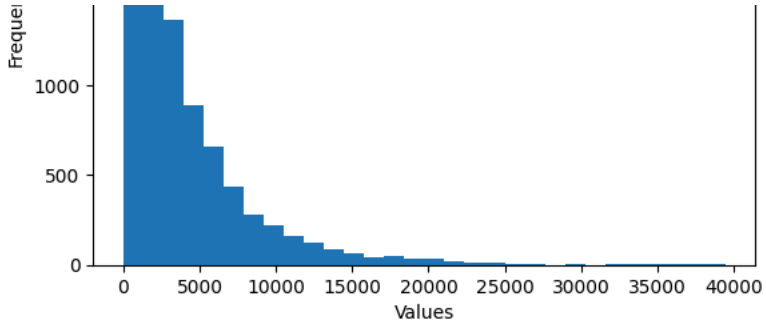

Awards Received
7000
8000

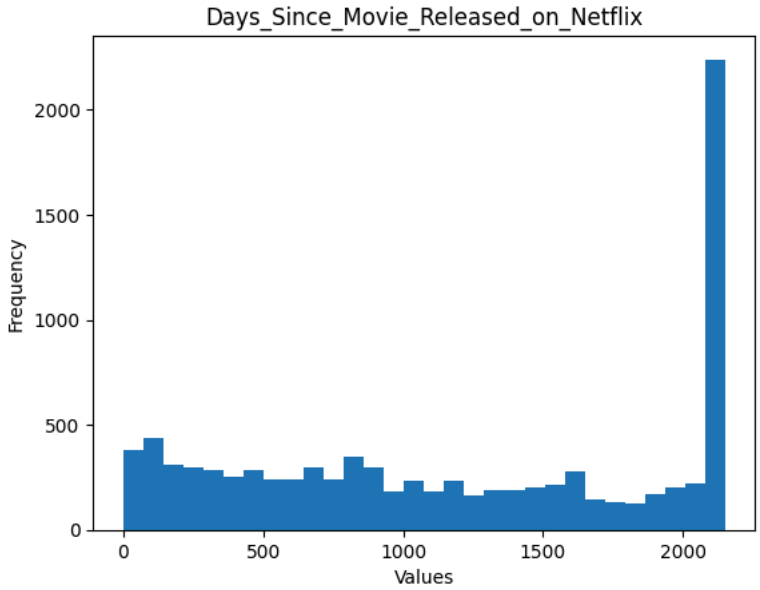

Frequency
Values
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200
250
300

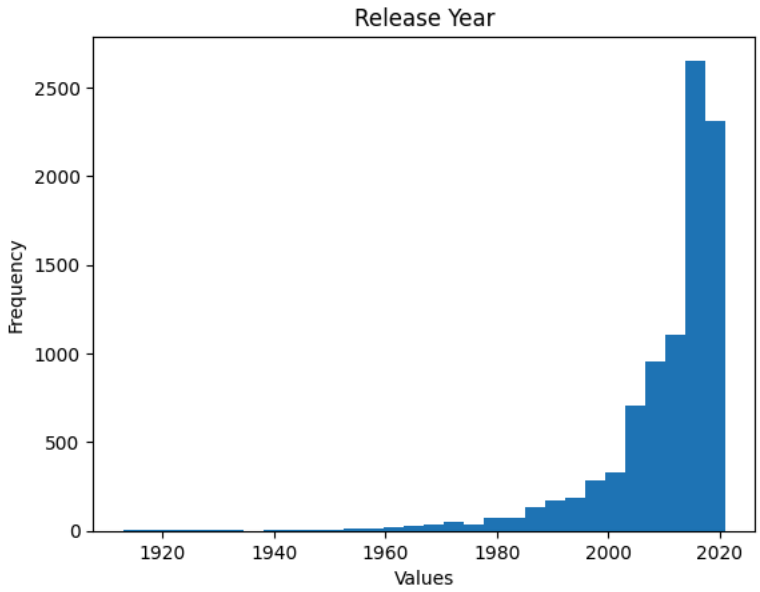

Awards Nominated For
Frequency
Values
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250
300
350
400

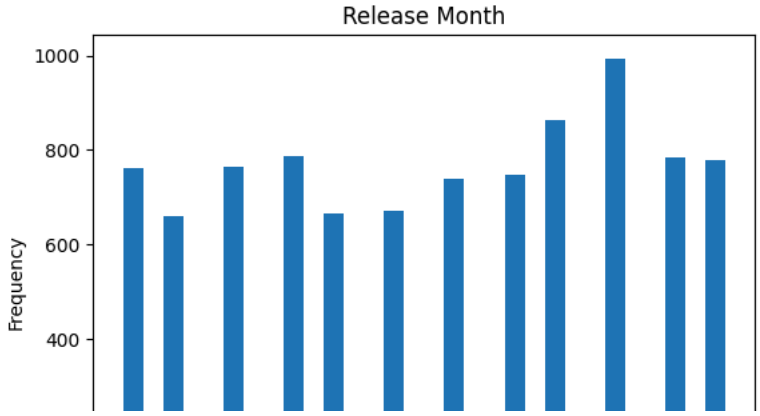

IMDb Votes
Frequency
Values
1e6
0
1000
2000
3000
4000
5000
6000
7000
0.0
0.5
1.0
1.5
2.0

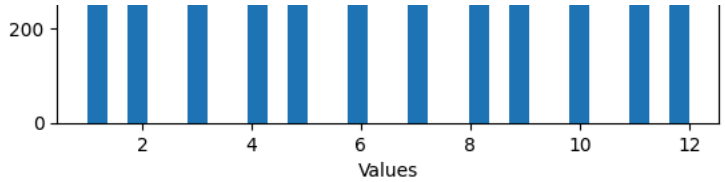

Days_Since_Movie_Released
2500
2000
1500


Freque
1000
500
0
0
5000
10000
15000
20000
25000
30000
35000
40000
Values


Days_Since_Movie_Released_on_Netflix
Frequency
2000
1500
1000
500
0
0
500
1000
1500
2000
Values


Release Year
Frequency
2500
2000
1500
1000
500
0
1920
1940
1960
1980
2000
2020
Values


Release Month
Frequency
1000
800
600
400




```
Pearson's First Coefficient of Skewness for Hidden Gem Score: -0.006706672020348805
Pearson's First Coefficient of Skewness for IMDb Score: -0.6414454765727965
Pearson's First Coefficient of Skewness for Rotten Tomatoes Score: -0.8508410923195496
Pearson's First Coefficient of Skewness for Awards Received: 7.473282616099965
Pearson's First Coefficient of Skewness for Awards Nominated For: 5.810237735721319
Pearson's First Coefficient of Skewness for IMDb Votes: 5.526193276102253
Pearson's First Coefficient of Skewness for Days_Since_Movie_Released: 2.7643856653018335
Pearson's First Coefficient of Skewness for Days_Since_Movie_Released_on_Netflix: -0.04782662642652016
Pearson's First Coefficient of Skewness for Release Year: -2.7610321481500946
Pearson's First Coefficient of Skewness for Release Month: -0.10273635829957133
```

Data is not normally distributed, so log transformation is applied stabilize the variance and reduce the impact of outliers.

```
# import numpy as np

# # Assuming your data is stored in a DataFrame called 'df'
# # Numeric columns
# numerical_columns = ["Hidden Gem Score", "IMDb Score", "Rotten Tomatoes Score",
#                      "Awards Received", "Awards Nominated For", "IMDb Votes",
#                      "Days_Since_Movie_Released", "Release Year"]

# # Apply log transformation to columns with positive skewness
# log_transform_columns = ["Awards Received", "IMDb Votes",
#                          "Days_Since_Movie_Released", "Release Year"]

# for column in log_transform_columns:
#     # df[column] = np.log(df[column])

# # Apply square root transformation to column with negative skewness
# square_root_transform_column = "IMDb Score"
# df[square_root_transform_column] = np.sqrt(df[square_root_transform_column])

```

```
# # Calculate skewness before transformation
# skewness_before = df[numerical_columns].skew()

# # Apply log transformation to columns with positive skewness
# log_transform_columns = ["Days_Since_Movie_Released", "Release Year"]

# for column in log_transform_columns:
#     # Add a small positive constant (1e-6) to the data to avoid zero or negative values
#     df[column] = np.log(df[column] + 1e-6)
# #
# # Apply square root transformation to column with negative skewness
# square_root_transform_column = "IMDb Score"
# # Adding a small constant to handle any negative values (e.g., 1e-6)
# df[square_root_transform_column] = np.sqrt(df[square_root_transform_column] + 1e-6)

# # Calculate skewness after transformation
# skewness_after = df[numerical_columns].skew()

# # Display skewness before and after transformation
# print("Skewness Before Transformation:")
# print(skewness_before)

# print("\nSkewness After Transformation:")
# print(skewness_after)

# # Plot histograms before and after transformation
# plt.figure(figsize=(12, 6))
# for i, column in enumerate(numerical_columns, 1):
#     plt.subplot(2, 4, i)
#     plt.hist(df[column], bins=30)
#     plt.xlabel('Values')
#     plt.ylabel('Frequency')
#     plt.title(column)

# plt.tight_layout()
# plt.show()
```

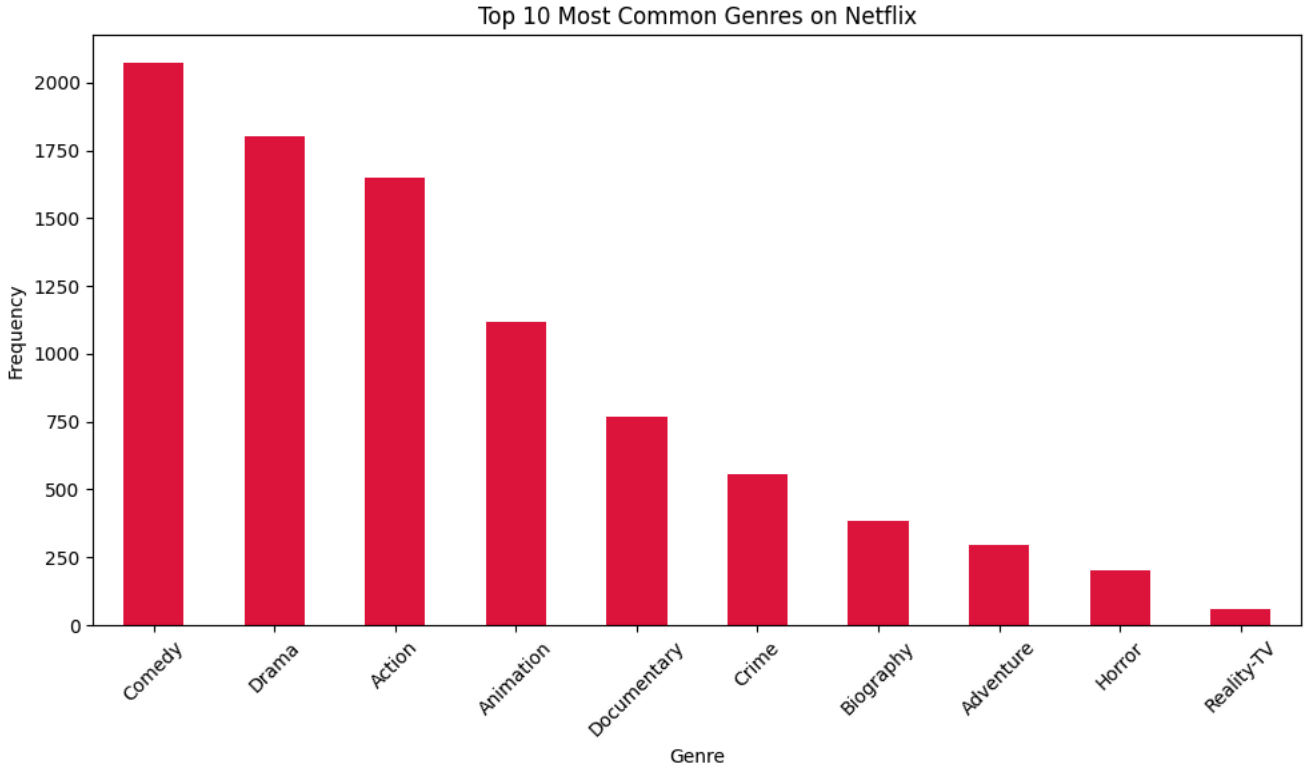
Descriptive analysis questions

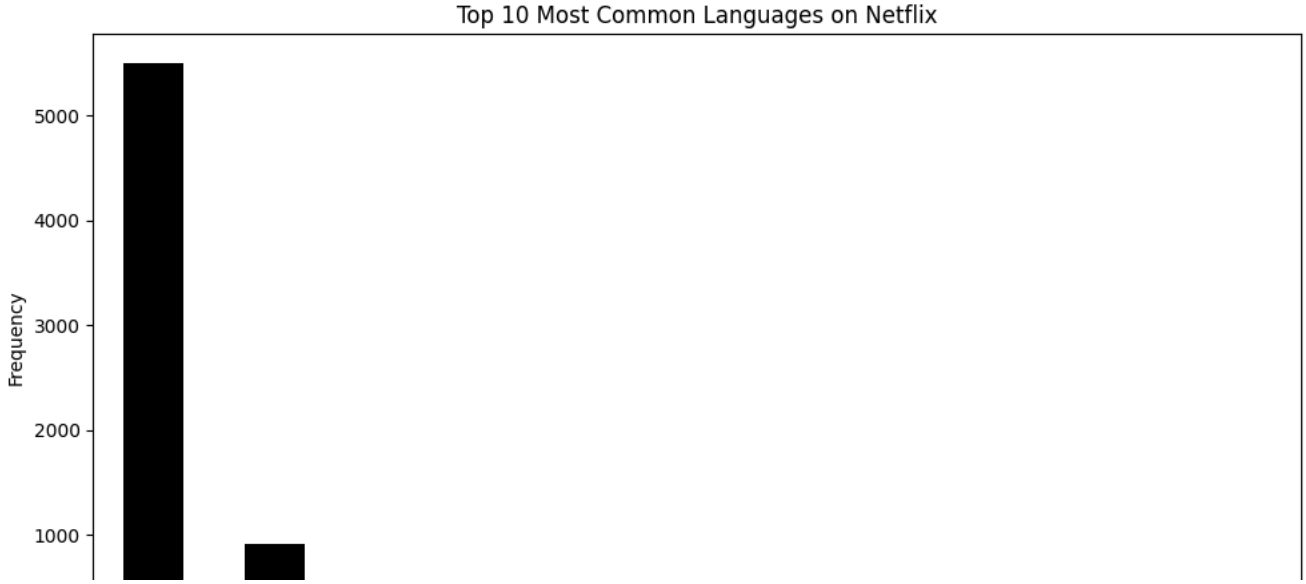
What are the most common genres and languages among movies and web series available on Netflix?

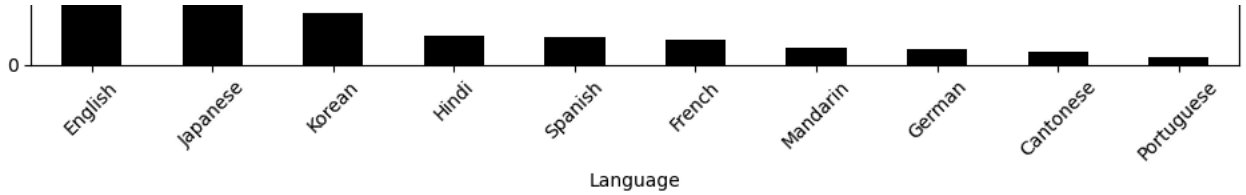
```

# Get the most common genres and languages
```

```
most_common_genres = df['Genre'].value_counts().head(10)
most_common_languages = df['Languages'].value_counts().head(10)

print("Most Common Genres:")
print(most_common_genres)

print("\nMost Common Languages:")
print(most_common_languages)

# Create bar chart for most common genres
plt.figure(figsize=(10, 6))
most_common_genres.plot(kind='bar', color='crimson')
plt.title('Top 10 Most Common Genres on Netflix')
plt.xlabel('Genre')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


# Create bar chart for most common languages
plt.figure(figsize=(10, 6))
most_common_languages.plot(kind='bar', color='black')
plt.title('Top 10 Most Common Languages on Netflix')
plt.xlabel('Language')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


plt.show()
```

```
Most Common Genres:
Comedy         2071
Drama          1801
Action         1651
Animation      1119
Documentary     767
Crime           556
Biography       386
Adventure       294
Horror          203
Reality-TV       57
Name: Genre, dtype: int64

Most Common Languages:
English       5498
Japanese       918
Korean         488
Hindi          278
Spanish        260
French         236
Mandarin       157
German         142
Cantonese      120
Portuguese      77
Name: Languages, dtype: int64
```



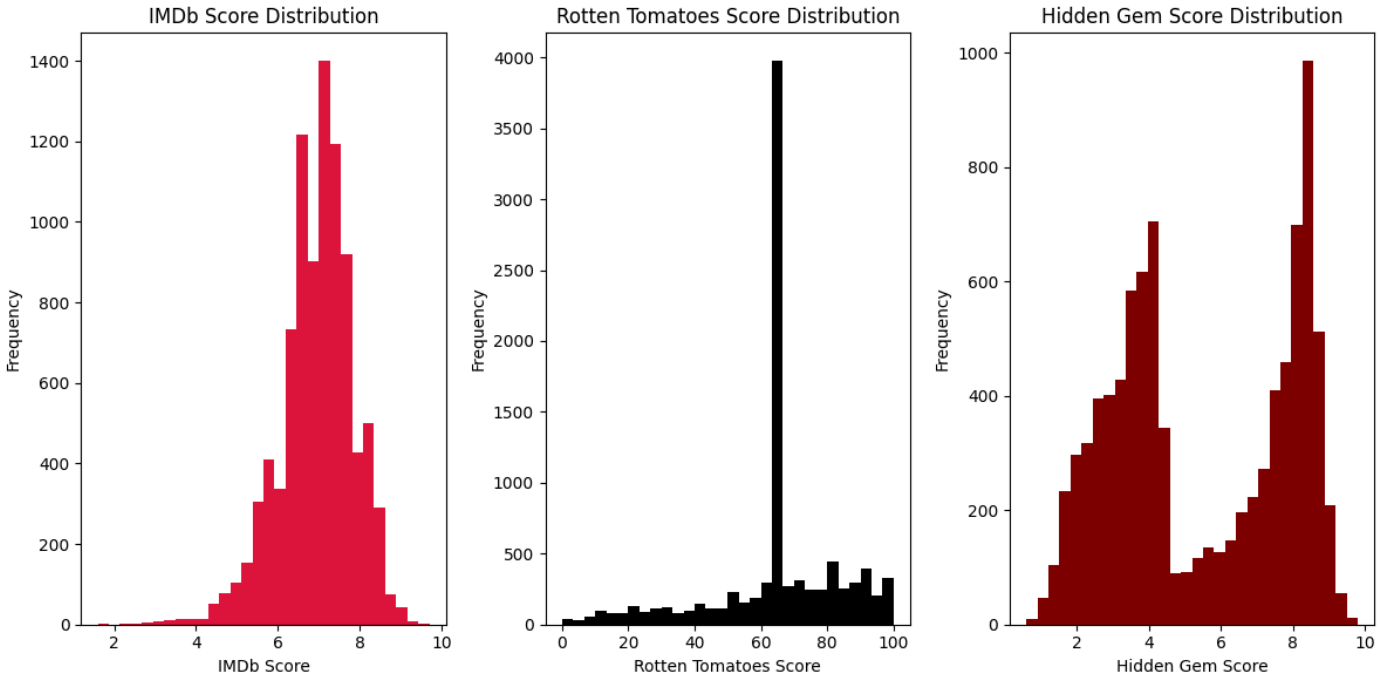





```
Start coding or generate with AI.
```

New Section

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

# IMDb Score distribution
plt.subplot(1, 3, 1)
plt.hist(df['IMDb Score'], bins=30, color='crimson')
plt.xlabel('IMDb Score')
plt.ylabel('Frequency')
plt.title('IMDb Score Distribution')

# Rotten Tomatoes Score distribution
plt.subplot(1, 3, 2)
plt.hist(df['Rotten Tomatoes Score'], bins=30, color='black')
plt.xlabel('Rotten Tomatoes Score')
plt.ylabel('Frequency')
plt.title('Rotten Tomatoes Score Distribution')

# Hidden Gem Score distribution
plt.subplot(1, 3, 3)
plt.hist(df['Hidden Gem Score'], bins=30, color='maroon')
plt.xlabel('Hidden Gem Score')
plt.ylabel('Frequency')
plt.title('Hidden Gem Score Distribution')

plt.tight_layout()
plt.show()

```



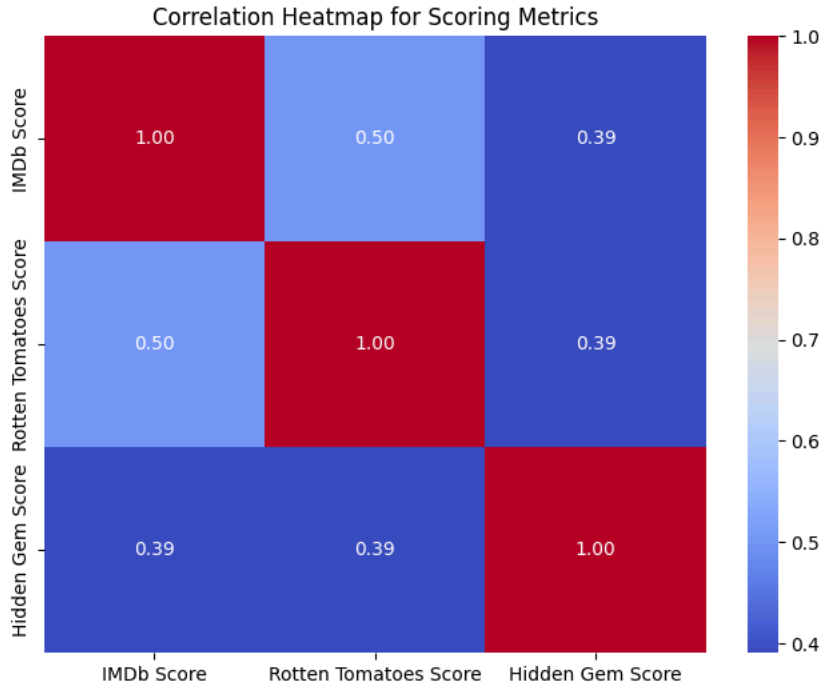

```
import scipy.stats as stats

# Assuming your data is stored in a DataFrame called 'df'
# Assuming 'IMDb Score', 'Rotten Tomatoes Score', and 'Hidden Gem Score' are the columns with scores

# Calculate the skewness of IMDb Score, Rotten Tomatoes Score, and Hidden Gem Score
imdb_skewness = stats.skew(df['IMDb Score'])
rt_skewness = stats.skew(df['Rotten Tomatoes Score'])
hg_skewness = stats.skew(df['Hidden Gem Score'])

print("Skewness of IMDb Score:", imdb_skewness)
print("Skewness of Rotten Tomatoes Score:", rt_skewness)
print("Skewness of Hidden Gem Score:", hg_skewness)
```

```
Skewness of IMDb Score: -0.6414454765727965
Skewness of Rotten Tomatoes Score: -0.8508410923195496
Skewness of Hidden Gem Score: -0.006706672020348805
```

Correlation between scores

```
# Select the relevant columns
scores_df = df[['IMDb Score', 'Rotten Tomatoes Score', 'Hidden Gem Score']]

# Calculate the correlation matrix
correlation_matrix = scores_df.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Visualize the correlation matrix as a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap for Scoring Metrics")
plt.show()
```

```
Correlation Matrix:
                      IMDb Score  Rotten Tomatoes Score  Hidden Gem Score
IMDb Score              1.000000               0.502980          0.390769
Rotten Tomatoes Score   0.502980               1.000000          0.390403
Hidden Gem Score        0.390769               0.390403          1.000000
```




```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 20 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Title                                 9217 non-null   object
 1   Genre                                 9217 non-null   object
 2   Tags                                  9217 non-null   object
 3   Languages                             9217 non-null   object
 4   Series or Movie                       9217 non-null   object
 5   Hidden Gem Score                      9217 non-null   float64
 6   Country Availability                  9217 non-null   object
 7   Runtime                               9217 non-null   object
 8   Director                              9217 non-null   object
 9   Writer                                9217 non-null   object
 10  Actors                                9217 non-null   object
 11  IMDb Score                            9217 non-null   float64
 12  Rotten Tomatoes Score                 9217 non-null   float64
 13  Awards Received                       9217 non-null   float64
 14  Awards Nominated For                  9217 non-null   float64
 15  IMDb Votes                            9217 non-null   float64
 16  Days_Since_Movie_Released             9217 non-null   int64
 17  Release Year                          9217 non-null   int64
 18  Release Month                         9217 non-null   int64
 19  Days_Since_Movie_Released_on_Netflix  9217 non-null   int64
dtypes: float64(6), int64(4), object(10)
memory usage: 1.5+ MB
```

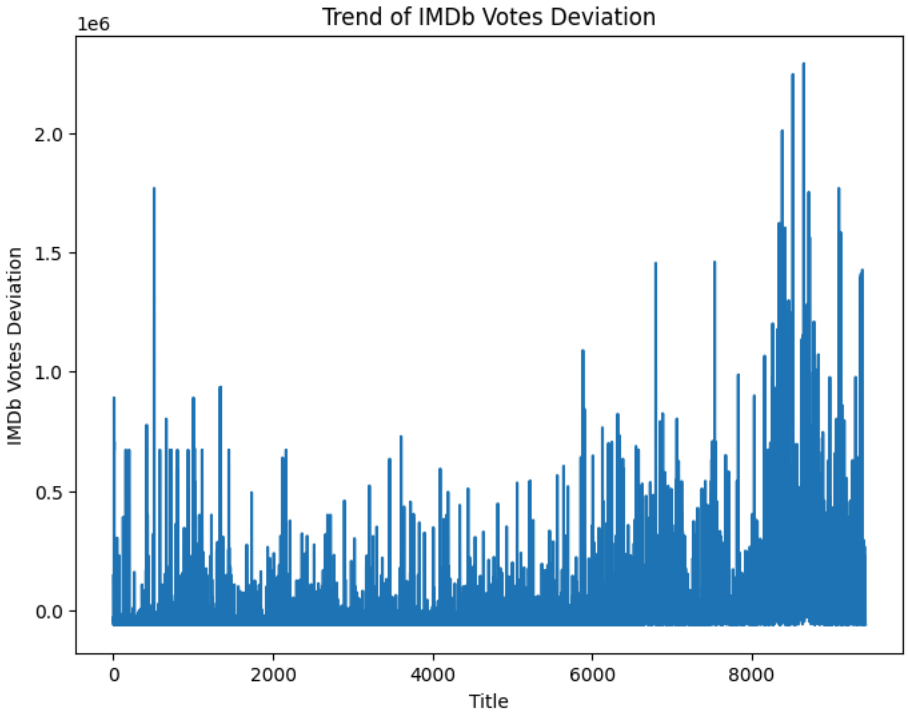
```
1 df.to_csv('/content/Final_Netflix_Data_For_Dashboard.csv')
```

Exploratory data analysis

```
1 import pandas as pd
2
3 # Assuming your data is stored in a DataFrame called 'df'
4 # Assuming 'IMDb Votes' is the relevant column
5
6 # Step 1: Calculate the average IMDb votes
7 avg_imdb_votes = df['IMDb Votes'].mean()
8
```

```
# Step 2: Create the new column by subtracting each IMDb vote record from the average IMDb votes
df['IMDb Votes Deviation'] = df['IMDb Votes'] - avg_imdb_votes

# Step 3: Examine the trend in the new column
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.plot(df['IMDb Votes Deviation'])
plt.xlabel('Title')
plt.ylabel('IMDb Votes Deviation')
plt.title('Trend of IMDb Votes Deviation')
plt.show()

```

```
<ipython-input-33-cc3ac611d136>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['IMDb Votes Deviation'] = df['IMDb Votes'] - avg_imdb_votes
```




```
df.head(2).transpose()
```

	0	1
Title	Lets Fight Ghost	HOW TO BUILD A GIRL
Genre	Crime	Comedy
Tags	Comedy Programmes	Dramas
Languages	Swedish	English
Series or Movie	Series	Movie
Hidden Gem Score	4.3	7.0
Country Availability	Thailand	Canada
Runtime	< 30 minutes	1-2 hour
Director	Tomas Alfredson	Coky Giedroyc
Writer	John Ajvide Lindqvist	Caitlin Moran
Actors	Lina Leandersson	Cleo
IMDb Score	7.9	5.8
Rotten Tomatoes Score	98.0	79.0
Awards Received	74.0	1.0
Awards Nominated For	57.0	0.0
IMDb Votes	205926.0	2838.0
Days_Since_Movie_Released	4718	553
Release Year	2008	2020
Release Month	12	5
Days_Since_Movie_Released_on_Netflix	0	0
IMDb Votes Deviation	144509.472088	-58578.527912

```
import pandas as pd


# Assuming your data is stored in a DataFrame called 'df'
# Assuming 'IMDb Score' and 'IMDb Votes' are the relevant columns

# Step 1: Calculate the average IMDb votes
avg_imdb_votes = df['IMDb Votes'].mean()

# Step 2: Create the IMDb Votes Deviation column
df['IMDb Votes Deviation'] = (df['IMDb Votes'] - avg_imdb_votes)/len(df['IMDb Votes'])

# Step 3: Define a function to calculate the weighted IMDb Score
def weighted_imdb_score(row):
    weight_imdb_votes = row['IMDb Votes Deviation']
    return (row['IMDb Score'] * (1 + weight_imdb_votes))

# Step 4: Apply the weighted_imdb_score function to create the weighted IMDb Score column
df['Weighted IMDb Score'] = df.apply(weighted_imdb_score, axis=1)

# Display the DataFrame with the new weighted IMDb Score column
print(df[['IMDb Score', 'IMDb Votes', 'Weighted IMDb Score']].head())

```

```
   IMDb Score  IMDb Votes  Weighted IMDb Score
0         7.9    205926.0           131.760782
1         5.8      2838.0           -31.061827
2         7.4       131.0           -41.803961
3         7.5        47.0           -42.437231
4         6.7        88.0           -37.880790
<ipython-input-35-c2d5213326ba>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['IMDb Votes Deviation'] = (df['IMDb Votes'] - avg_imdb_votes)/len(df['IMDb Votes'])
<ipython-input-35-c2d5213326ba>:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Weighted IMDb Score'] = df.apply(weighted_imdb_score, axis=1)
```

```
df['Weighted IMDb Score'].value_counts()
```

```
 480.350121    12
-39.666893      9
-13.883281      8
 126.057154     6
 323.660372     6
               ..
-40.385250      1
-40.431063      1
-36.042922      1
-43.386771      1
-44.027245      1
Name: Weighted IMDb Score, Length: 8701, dtype: int64
```

```
# Step 5: Calculate the correlation between 'Awards Nominated For' and 'Weighted IMDb Score' for 'Hit' titles
correlation = df['Awards Nominated For'].corr(df['Weighted IMDb Score'])

print("Correlation between 'Awards Nominated For' and 'Weighted IMDb Score' for 'Hit' titles:", correlation)
```

```
Correlation between 'Awards Nominated For' and 'Weighted IMDb Score' for 'Hit' titles: 0.5090434323505878
```

```
# # Step 1: Calculate the 99th percentile of 'Awards Nominated For'
# percentile_95 = np.percentile(df['Awards Nominated For'], 95)

# # Step 2: Remove values greater than 100
# df = df[df['Awards Nominated For'] <= percentile_95]


```

```
df['Awards Nominated For'].value_counts()
```

```
0.0      2895
1.0      1006
2.0       694
3.0       534
4.0       421
         ...
248.0       1
165.0       1
112.0       1
263.0       1
138.0       1
Name: Awards Nominated For, Length: 201, dtype: int64
```

```
df['Weighted IMDb Score'].value_counts()
```

```
 480.350121    12
-39.666893      9
-13.883281      8
 126.057154     6
 323.660372     6
               ..
-40.385250      1
-40.431063      1
-36.042922      1
-43.386771      1
-44.027245      1
Name: Weighted IMDb Score, Length: 8701, dtype: int64
```

```
positive_count = df[df['Weighted IMDb Score'] > threshold].shape[0]
negative_count = df[df['Weighted IMDb Score'] <= threshold].shape[0]

print("Number of Positive Values:", positive_count)
print("Number of Negative Values:", negative_count)
```

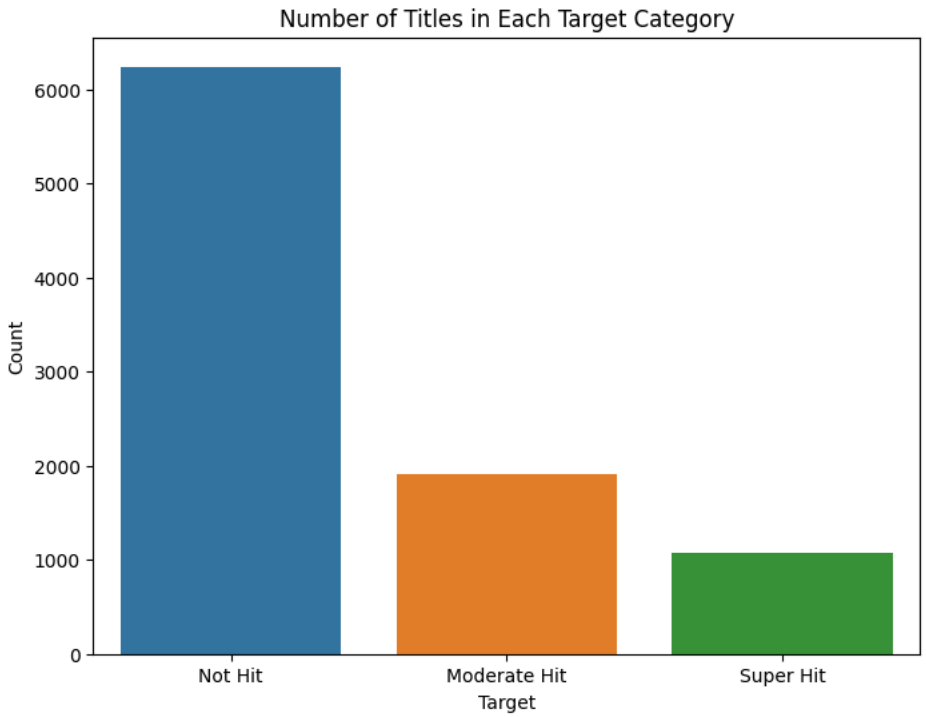
```
Number of Positive Values: 134
Number of Negative Values: 9083
```

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np

# Assuming your data is stored in a DataFrame called 'df'
# Assuming 'Awards Nominated For', 'Weighted IMDb Score', and 'Target' are relevant columns

# Step 1: Calculate the 80th percentile of 'Awards Nominated For'
percentile_80 = np.percentile(df['Awards Nominated For'], 80)
print('vscdfgh',percentile_80)

percentile_70 = np.percentile(df['Awards Nominated For'], 60)
print('70%',percentile_70)

# Step 2: Define the success_label function
def success_label(nominated_for, weighted_imdb_score, threshold,percentile_70):
    if nominated_for > threshold and weighted_imdb_score > 0:
        return 'Super Hit'
    elif nominated_for > threshold or weighted_imdb_score > 0:
         return 'Moderate Hit'
    else:
        return 'Not Hit'

# Step 3: Apply the success_label function to create the 'Target' variable
df['Target'] = df.apply(lambda row: success_label(row['Awards Nominated For'],
                                                  row['Weighted IMDb Score'],
                                                  percentile_80,percentile_70), axis=1)

# Check the value counts of the 'Target' variable to ensure it's balanced
print(df['Target'].value_counts())

```

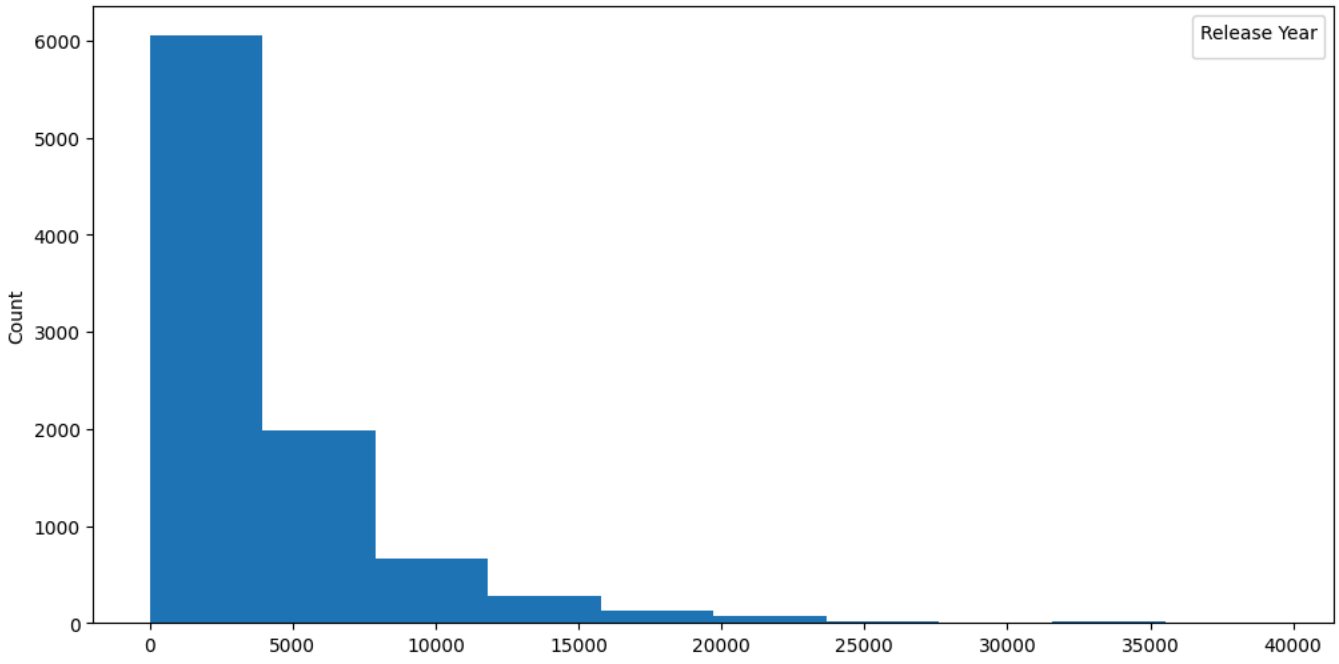
```
vscdfgh 13.0
70% 4.0
Not Hit         6241
Moderate Hit    1909
Super Hit       1067
Name: Target, dtype: int64
<ipython-input-42-40b46964ec3c>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
  df['Target'] = df.apply(lambda row: success_label(row['Awards Nominated For'],
```

```
# Calculate value counts for the 'Target' variable
target_value_counts = df['Target'].value_counts()

# Create a bar chart for the 'Target' variable
plt.figure(figsize=(8, 6))
sns.barplot(x=target_value_counts.index, y=target_value_counts.values)
plt.xlabel('Target')
plt.ylabel('Count')
plt.title('Number of Titles in Each Target Category')
plt.show()
```



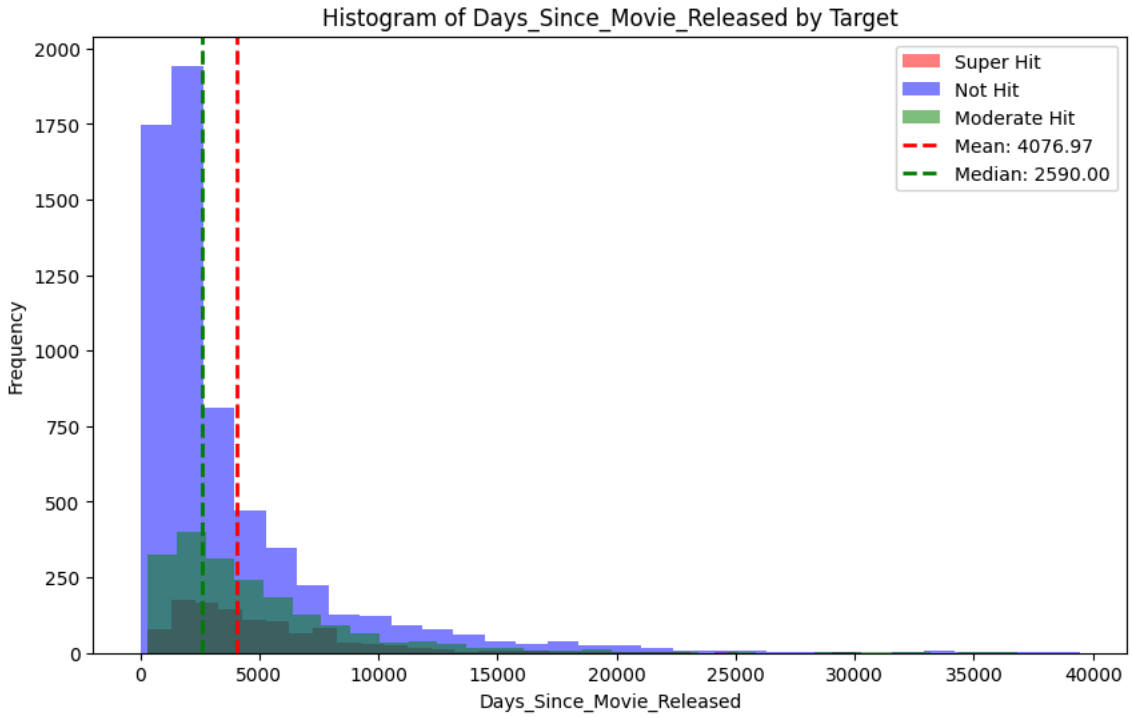

```
plt.figure(figsize=(12, 6))
plt.hist(x='Days_Since_Movie_Released', data=df, bins=10)
# plt.xlabel('Release Year')
plt.ylabel('Count')
# plt.title('Count Plot of Release Year with Target as the Legend')
plt.legend(title='Release Year', loc='upper right')  # Customize the legend title and location
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an unders
```




```
# Assuming you have a DataFrame named 'df' and want to plot a histogram of a column 'data_column'
data_column = 'Days_Since_Movie_Released'

categorical_column = 'Target'  # Replace with the name of your categorical column

# Get unique categories in the categorical column and assign colors to each category
unique_categories = df[categorical_column].unique()
custom_colors = ['red', 'blue', 'green']  # Add more colors as needed
```

```
category_colors = custom_colors[:len(unique_categories)]

plt.figure(figsize=(10, 6))
for i, category in enumerate(unique_categories):
    category_data = df[df[categorical_column] == category][data_column]
    plt.hist(category_data, bins=30, alpha=0.5, color=category_colors[i], label=category)

# Calculate mean and median for the overall data
mean_val = df[data_column].mean()
median_val = df[data_column].median()

# Add mean and median lines to the histogram
plt.axvline(mean_val, color='red', linestyle='dashed', linewidth=2, label=f'Mean: {mean_val:.2f}')
plt.axvline(median_val, color='green', linestyle='dashed', linewidth=2, label=f'Median: {median_val:.2f}')

plt.xlabel(data_column)
plt.ylabel('Frequency')
plt.title(f'Histogram of {data_column} by {categorical_column}')
plt.legend()
plt.show()
```



Cluster analysis

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 23 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Title                           9217 non-null   object
 1   Genre                           9217 non-null   object
 2   Tags                            9217 non-null   object
 3   Languages                       9217 non-null   object
 4   Series or Movie                 9217 non-null   object
 5   Hidden Gem Score                9217 non-null   float64
 6   Country Availability            9217 non-null   object
 7   Runtime                         9217 non-null   object
 8   Director                        9217 non-null   object
 9   Writer                          9217 non-null   object
 10  Actors                          9217 non-null   object
 11  IMDb Score                      9217 non-null   float64
 12  Rotten Tomatoes Score           9217 non-null   float64
 13  Awards Received                 9217 non-null   float64
 14  Awards Nominated For            9217 non-null   float64
 15  IMDb Votes                      9217 non-null   float64
 16  Days_Since_Movie_Released       9217 non-null   int64
```

```
 17  Release Year                          9217 non-null   int64
 18  Release Month                         9217 non-null   int64
 19  Days_Since_Movie_Released_on_Netflix  9217 non-null   int64
 20  IMDb Votes Deviation                  9217 non-null   float64
 21  Weighted IMDb Score                   9217 non-null   float64
 22  Target                                9217 non-null   object
dtypes: float64(8), int64(4), object(11)
memory usage: 1.7+ MB
```

```
df_copy = df.copy()
```

```
df = df_copy.copy()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 23 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Title                                 9217 non-null   object
 1   Genre                                 9217 non-null   object
 2   Tags                                  9217 non-null   object
 3   Languages                             9217 non-null   object
 4   Series or Movie                       9217 non-null   object
 5   Hidden Gem Score                      9217 non-null   float64
 6   Country Availability                  9217 non-null   object
 7   Runtime                               9217 non-null   object
 8   Director                              9217 non-null   object
 9   Writer                                9217 non-null   object
 10  Actors                                9217 non-null   object
 11  IMDb Score                            9217 non-null   float64
 12  Rotten Tomatoes Score                 9217 non-null   float64
 13  Awards Received                       9217 non-null   float64
 14  Awards Nominated For                  9217 non-null   float64
 15  IMDb Votes                            9217 non-null   float64
 16  Days_Since_Movie_Released             9217 non-null   int64
 17  Release Year                          9217 non-null   int64
 18  Release Month                         9217 non-null   int64
 19  Days_Since_Movie_Released_on_Netflix  9217 non-null   int64
 20  IMDb Votes Deviation                  9217 non-null   float64
 21  Weighted IMDb Score                   9217 non-null   float64
 22  Target                                9217 non-null   object
dtypes: float64(8), int64(4), object(11)
memory usage: 1.7+ MB
```

```
df_cluster = df.drop(['Genre', 'Languages', 'Series or Movie', 'Country Availability', 'Runtime',
                      'Director', 'Writer', 'Actors','Tags','Title','Target'], axis=1)
```

```
df_cluster.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 12 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Hidden Gem Score                      9217 non-null   float64
 1   IMDb Score                            9217 non-null   float64
 2   Rotten Tomatoes Score                 9217 non-null   float64
 3   Awards Received                       9217 non-null   float64
 4   Awards Nominated For                  9217 non-null   float64
 5   IMDb Votes                            9217 non-null   float64
 6   Days_Since_Movie_Released             9217 non-null   int64
 7   Release Year                          9217 non-null   int64
 8   Release Month                         9217 non-null   int64
 9   Days_Since_Movie_Released_on_Netflix  9217 non-null   int64
 10  IMDb Votes Deviation                  9217 non-null   float64
 11  Weighted IMDb Score                   9217 non-null   float64
dtypes: float64(8), int64(4)
memory usage: 936.1 KB
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Drop the columns that are not suitable for clustering (e.g., non-numeric columns)


# Standardize the numerical features
```

```
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_cluster)

# Apply K-Means clustering with k=3
kmeans = KMeans(n_clusters=3, random_state=42)
cluster_labels = kmeans.fit_predict(df_scaled)

# Add cluster labels back to the DataFrame
df['Cluster'] = cluster_labels

# Get the cluster centroids
cluster_centroids = pd.DataFrame(kmeans.cluster_centers_, columns=df_cluster.columns)

# Analyze the cluster centroids to identify important features
print("Cluster Centroids:")
# print(cluster_centroids)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
Cluster Centroids:
```

```
cluster_centroids
```

	Hidden Gem Score	IMDb Score	Rotten Tomatoes Score	Awards Received	Awards Nominated For	IMDb Votes	Days_Since_Movie_Released	Release Year	Release Month	Days_Since_Movie_R
0	-0.912693	-0.556360	-0.633800	-0.100028	-0.080789	0.171243	0.676031	-0.675439	-0.009020	
1	0.532708	0.225876	0.269915	-0.150893	-0.182679	-0.338700	-0.383218	0.383205	-0.008167	
2	-0.615271	0.964709	0.928227	2.768927	3.060504	3.351275	0.301794	-0.305903	0.175956	

```
Start coding or generate with AI.
```

```
# Look at the distribution of features within each cluster
for i in range(3):
    print(f"Cluster {i}:")
    print(df[df['Cluster'] == i].describe(include='all'))
```

freq	NaN	NaN		NaN
mean	2006.602837	7.304965		1678.959811
std	10.276887	3.593589		713.728277
min	1960.000000	1.000000		0.000000
25%	2002.000000	5.000000		1374.000000
50%	2009.000000	7.000000		2151.000000
75%	2014.000000	11.000000		2151.000000
max	2020.000000	12.000000		2151.000000

	IMDb Votes Deviation	Weighted IMDb Score	Target	Cluster
count	423.000000	423.000000	423	423.0
unique	NaN	NaN	2	NaN
top	NaN	NaN	Super Hit	NaN
freq	NaN	NaN	399	NaN
mean	53.696806	436.581293	NaN	2.0
std	37.770202	332.925967	NaN	0.0
min	-5.351473	-35.246932	NaN	2.0
25%	29.746010	235.826651	NaN	2.0
50%	46.471571	359.182037	NaN	2.0
75%	68.679665	511.312742	NaN	2.0
max	248.755612	2322.727188	NaN	2.0

[11 rows x 24 columns]

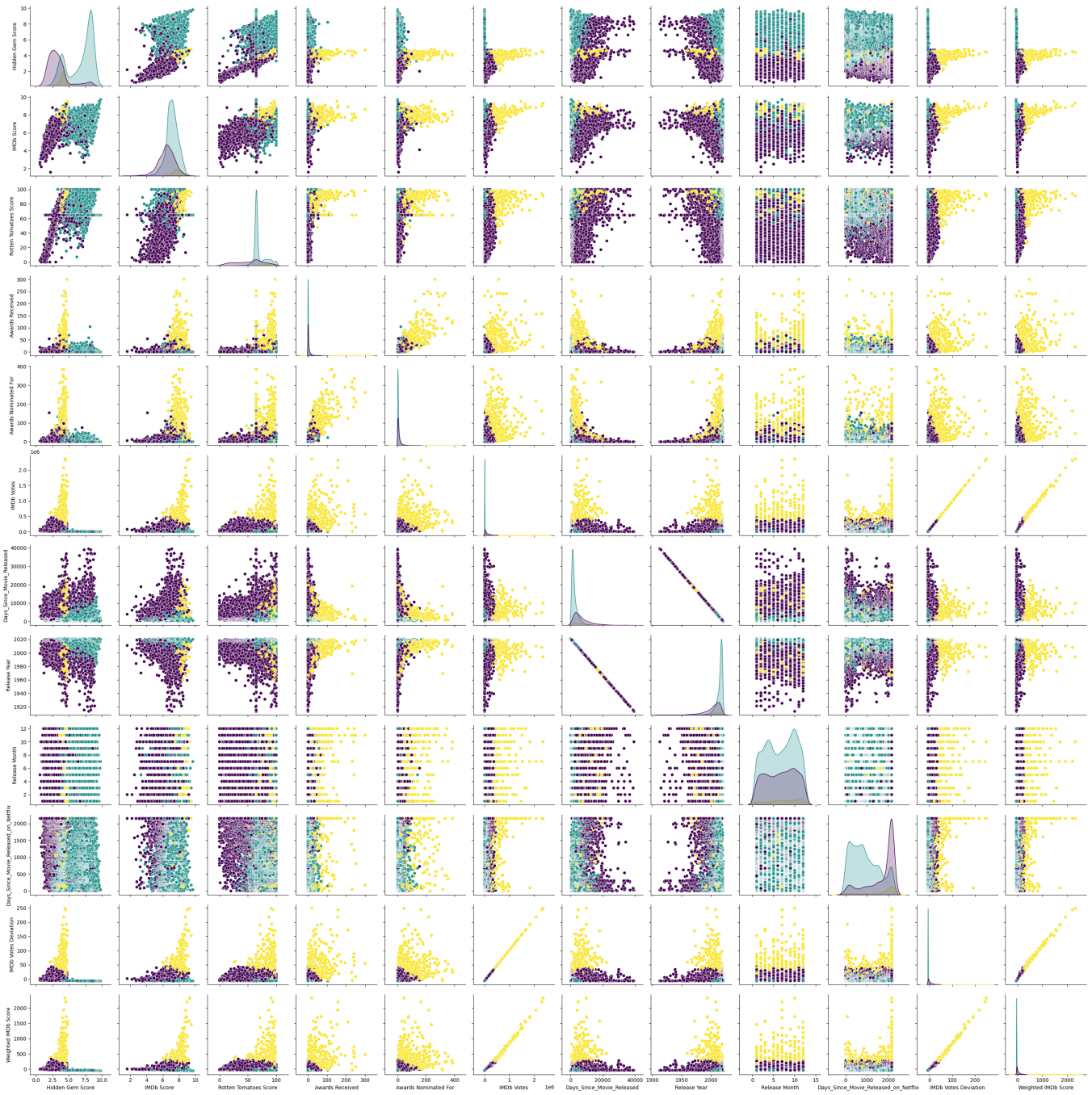
```
df.drop(['Title'],axis=1,inplace=True)
```

```
# Analyze each cluster
for i in range(df['Cluster'].nunique()):
    print(f"Cluster {i}:")

    # Identify Patterns and Similarities
    print("Common properties:")
    print(df[df['Cluster'] == i].mode().iloc[0])  # prints the mode of each feature within the cluster

    # Feature Importance
    # In KMeans, all features contribute equally to the distance calculation. However, you can look at the mode or mean value

    # Understanding Audience Segments
    print("Audience segment properties:")
    df[df['Cluster'] == i].describe()  # prints summary statistics of each feature within the cluster
```

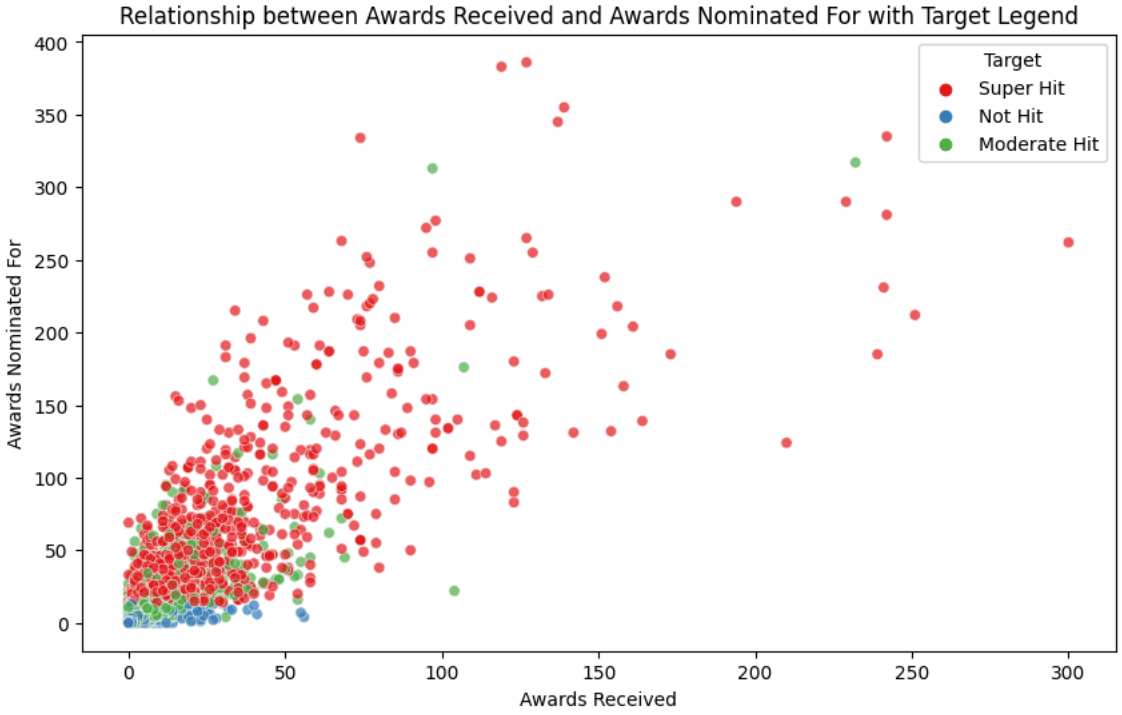
```
Target                                    Not Hit
Cluster                                         0
Name: 0, dtype: object
Audience segment properties:
Cluster 1:
Common properties:
Genre                                      Comedy
Tags                                       Dramas
Languages                                 English
Series or Movie                             Movie
Hidden Gem Score                              8.3
Country Availability                        Japan
Runtime                                   1-2 hour
Director                                  Unknown
Writer                                    Unknown
Actors                                Jackie Chan
IMDb Score                                    6.8
Rotten Tomatoes Score                   64.691276
```

```
Writer                                  George Lucas
Actors                                 Ewan McGregor
IMDb Score                                       7.6
Rotten Tomatoes Score                      64.691276
Awards Received                                 26.0
Awards Nominated For                            69.0
IMDb Votes                                  733336.0
Days_Since_Movie_Released                       8213
Release Year                                    2013
Release Month                                     12
Days_Since_Movie_Released_on_Netflix            2151
IMDb Votes Deviation                       72.900019
Weighted IMDb Score                       480.350121
Target                                     Super Hit
Cluster                                            2
Name: 0, dtype: object
Audience segment properties:
```

```
# Visualize the clusters
sns.pairplot(df, hue='Cluster', diag_kind='kde', palette='viridis')
plt.show()
```

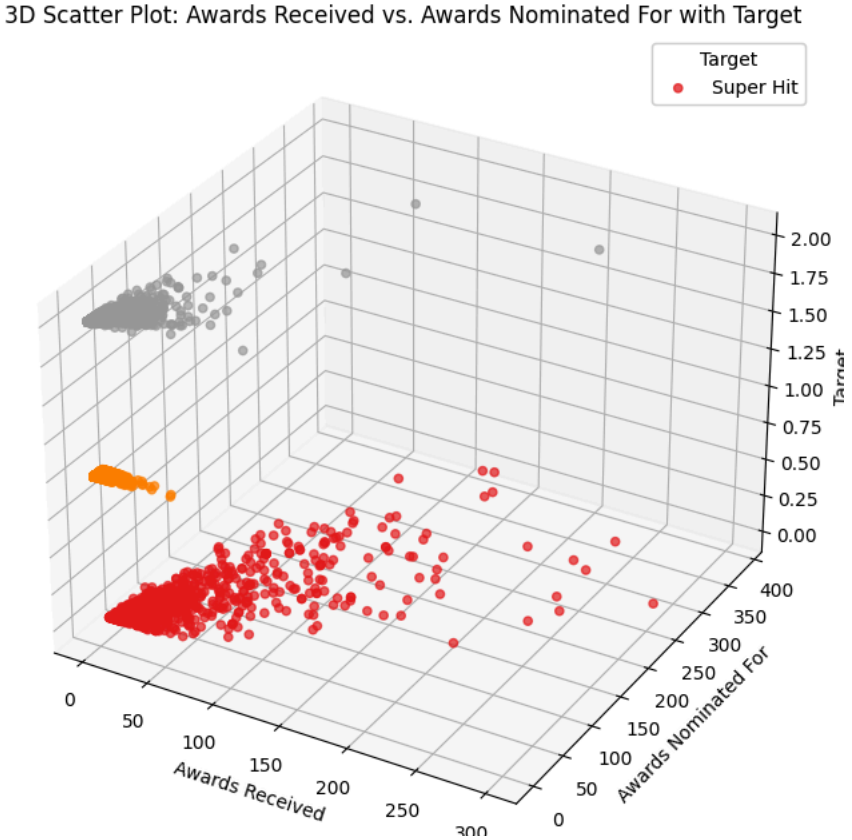

Hidden Gem Score
IMDb Score
Rotten Tomatoes Score
Awards Received
Awards Nominated For
IMDb Votes
Days_Since_Movie_Released
Release Year
Release Month
Days_Since_Movie_Released_on_Netflix
IMDb Votes Deviation
Weighted IMDb Score
Cluster
0
1
2

```
import seaborn as sns
import matplotlib.pyplot as plt

# Scatter plot with legend based on 'Target' variable
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Awards Received', y='Awards Nominated For', hue='Target', data=df, palette='Set1', alpha=0.7)
plt.xlabel('Awards Received')
plt.ylabel('Awards Nominated For')
plt.title('Relationship between Awards Received and Awards Nominated For with Target Legend')
plt.legend(title='Target', loc='upper right')
plt.show()

```




```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd

# Convert 'Target' column to numeric category codes
df['Target_code'] = pd.factorize(df['Target'])[0]

# Create a mapping of numeric category codes to 'Target' labels
target_mapping = {code: label for code, label in zip(df['Target_code'], df['Target'])}

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot with color based on 'Target' variable
scatter = ax.scatter(df['Awards Received'], df['Awards Nominated For'], df['Target_code'], c=df['Target_code'], cmap='Set1',

# Set labels for x, y, and z axes
ax.set_xlabel('Awards Received')
ax.set_ylabel('Awards Nominated For')
ax.set_zlabel('Target')

# Add legend with 'Target' labels
target_labels = [target_mapping[code] for code in sorted(df['Target_code'].unique())]
legend = ax.legend(*scatter.legend_elements(), labels=target_labels, title='Target', loc='upper right')
ax.add_artist(legend)

# Show the 3D scatter plot
plt.title('3D Scatter Plot: Awards Received vs. Awards Nominated For with Target')
plt.show()

```

```
<ipython-input-59-22f0547d105e>:25: UserWarning: You have mixed positional and keyword arguments, some input may be discarde
  legend = ax.legend(*scatter.legend_elements(), labels=target_labels, title='Target', loc='upper right')
```



Predictive analysis

Logistic regression with all columns

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9217 entries, 0 to 9423
Data columns (total 24 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Genre                               9217 non-null   object
 1   Tags                                9217 non-null   object
 2   Languages                           9217 non-null   object
 3   Series or Movie                     9217 non-null   object
 4   Hidden Gem Score                    9217 non-null   float64
 5   Country Availability                9217 non-null   object
 6   Runtime                             9217 non-null   object
 7   Director                            9217 non-null   object
 8   Writer                              9217 non-null   object
 9   Actors                              9217 non-null   object
 10  IMDb Score                          9217 non-null   float64
 11  Rotten Tomatoes Score               9217 non-null   float64
```