```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, precision_score,
recall_score

mov=pd.read_csv("movies.csv")
mov

mov.head()

   movie_id                          movie_name  \
0         1                    Toy Story (1995)
1         2                      Jumanji (1995)
2         3             Grumpier Old Men (1995)
3         4            Waiting to Exhale (1995)
4         5  Father of the Bride Part II (1995)

                                    movie_type
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy

mov.tail()

       movie_id                      movie_name
movie_type
62418    209157                       We (2018)
Drama
62419    209159       Window of the Soul (2001)
Documentary
62420    209163                Bad Poems (2018)            Comedy|
Drama
62421    209169             A Girl Thing (2001)      (no genres
listed)
62422    209171  Women of Devil's Island (1962)  Action|Adventure|
Drama

mov.index

RangeIndex(start=0, stop=62423, step=1)

mov.info
```

```
<bound method DataFrame.info of        movie_id
movie_name  \
0             1                    Toy Story (1995)
1             2                      Jumanji (1995)
2             3             Grumpier Old Men (1995)
3             4            Waiting to Exhale (1995)
4             5  Father of the Bride Part II (1995)
...         ...                                 ...
62418    209157                           We (2018)
62419    209159           Window of the Soul (2001)
62420    209163                    Bad Poems (2018)
62421    209169                 A Girl Thing (2001)
62422    209171      Women of Devil's Island (1962)

                                        movie_type
0      Adventure|Animation|Children|Comedy|Fantasy
1                       Adventure|Children|Fantasy
2                                   Comedy|Romance
3                             Comedy|Drama|Romance
4                                           Comedy
...                                            ...
62418                                        Drama
62419                                  Documentary
62420                                 Comedy|Drama
62421                           (no genres listed)
62422                       Action|Adventure|Drama

[62423 rows x 3 columns]>
```

```
mov.describe()
```

```
            movie_id
count   62423.000000
mean   122220.387646
std     63264.744844
min         1.000000
25%     82146.500000
50%    138022.000000
75%    173222.000000
max    209171.000000
```

```
mov.shape
```

```
(62423, 3)
```

```
mov.columns
```

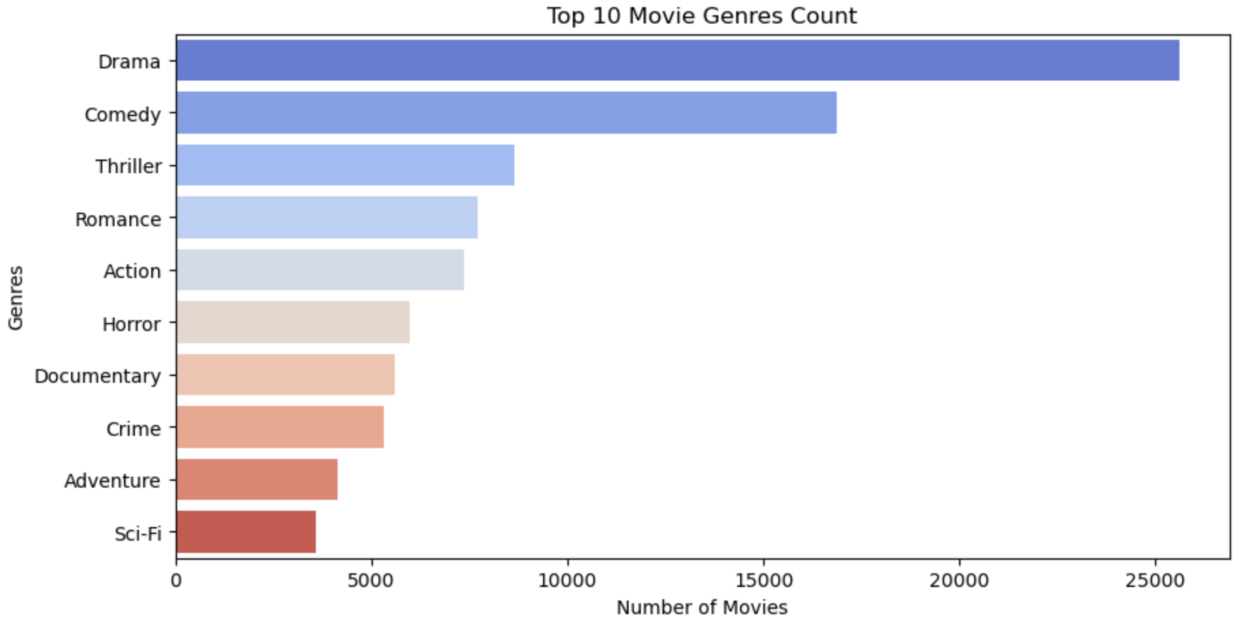
```
Index(['movie_id', 'movie_name', 'movie_type'], dtype='object')
```

```
print("Null data: ", mov.isna().sum())
print("Duplicate data: ", mov.duplicated().sum())
```

```
Null data:  movie_id      0
movie_name    0
movie_type    0
dtype: int64
Duplicate data:  0
```

```python
# Split and count movie genres
genre_counts = mov["movie_type"].str.split("|",
expand=True).stack().value_counts()

# Select the top 10 genres (excluding missing category)
top_10_genres = genre_counts.drop("(no genres listed)",
errors="ignore").head(10)

# Plot the corrected bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x=top_10_genres.values, y=top_10_genres.index,
palette="coolwarm")

# Labels and title
plt.xlabel("Number of Movies")
plt.ylabel("Genres")
plt.title("Top 10 Movie Genres Count")

# Show plot
plt.show()
```

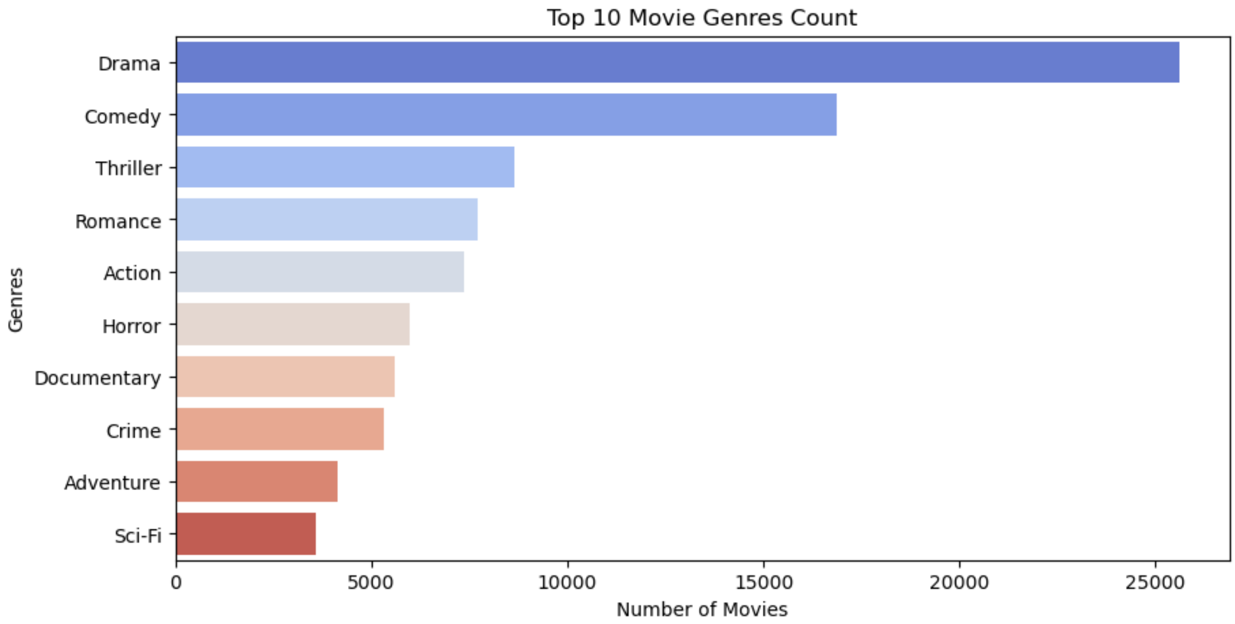
```
C:\Users\Mi\AppData\Local\Temp\ipykernel_13872\3566090564.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=top_10_genres.values, y=top_10_genres.index,
palette="coolwarm")
```



```
# Split and count movie genres
genre_counts = mov["movie_type"].str.split("|",
expand=True).stack().value_counts()

# Select the top 10 genres (excluding missing category)
top_10_genres = genre_counts.drop("(no genres listed)",
errors="ignore").head(10)

# Plot the corrected bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x=top_10_genres.values, y=top_10_genres.index,
palette="coolwarm")

# Labels and title
plt.xlabel("Number of Movies")
plt.ylabel("Genres")
plt.title("Top 10 Movie Genres Count")

# Show plot
plt.show()
```

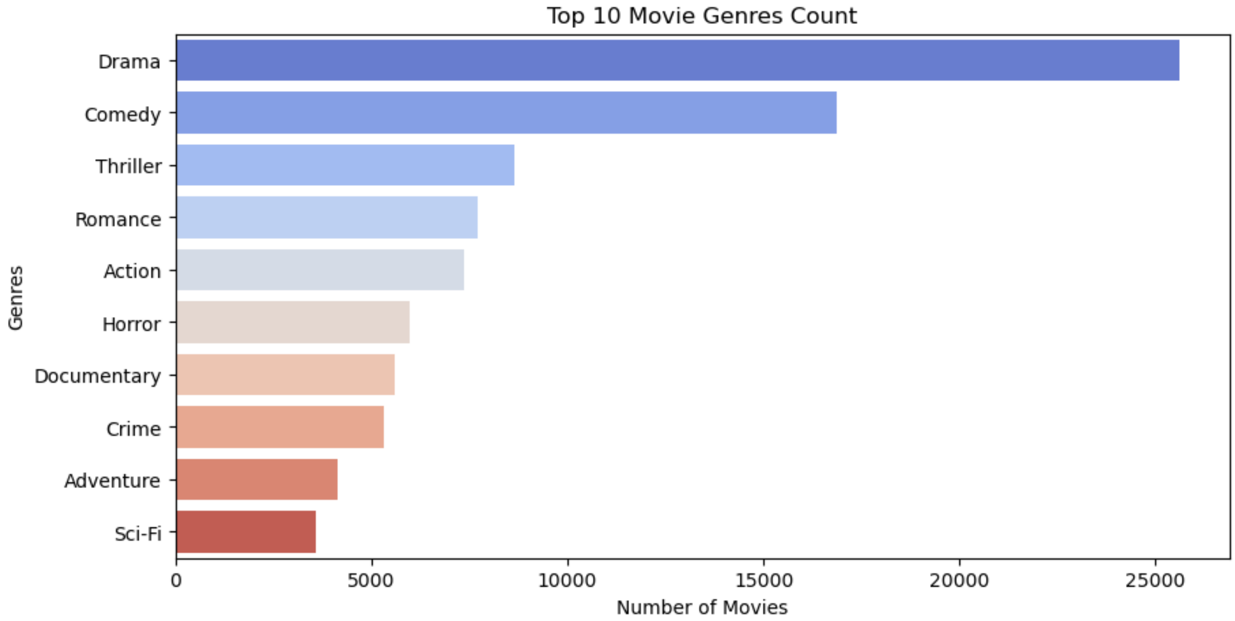
```
C:\Users\Mi\AppData\Local\Temp\ipykernel_13872\3566090564.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.
```

```
  sns.barplot(x=top_10_genres.values, y=top_10_genres.index,
palette="coolwarm")
```

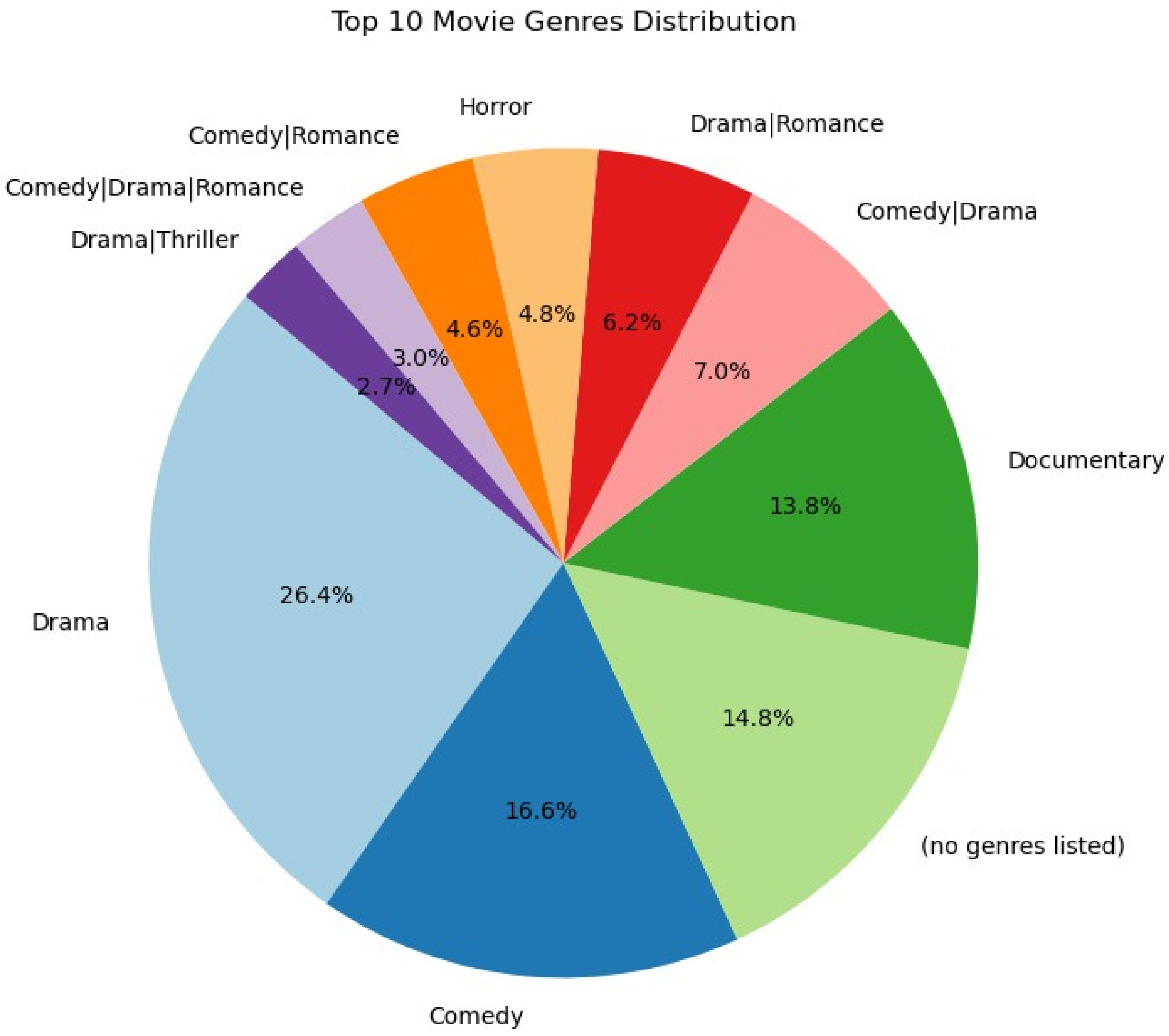


```
# Split and count movie genres
genre_counts = mov["movie_type"].str.split("|",
expand=True).stack().value_counts()

# Select the top 10 genres (excluding missing category)
top_10_genres = genre_counts.drop("(no genres listed)",
errors="ignore").head(10)

# Plot the corrected bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x=top_10_genres.values, y=top_10_genres.index,
palette="coolwarm")

# Labels and title
plt.xlabel("Number of Movies")
plt.ylabel("Genres")
plt.title("Top 10 Movie Genres Count")

# Show plot
plt.show()


C:\Users\Mi\AppData\Local\Temp\ipykernel_13872\3566090564.py:9:
FutureWarning:
```

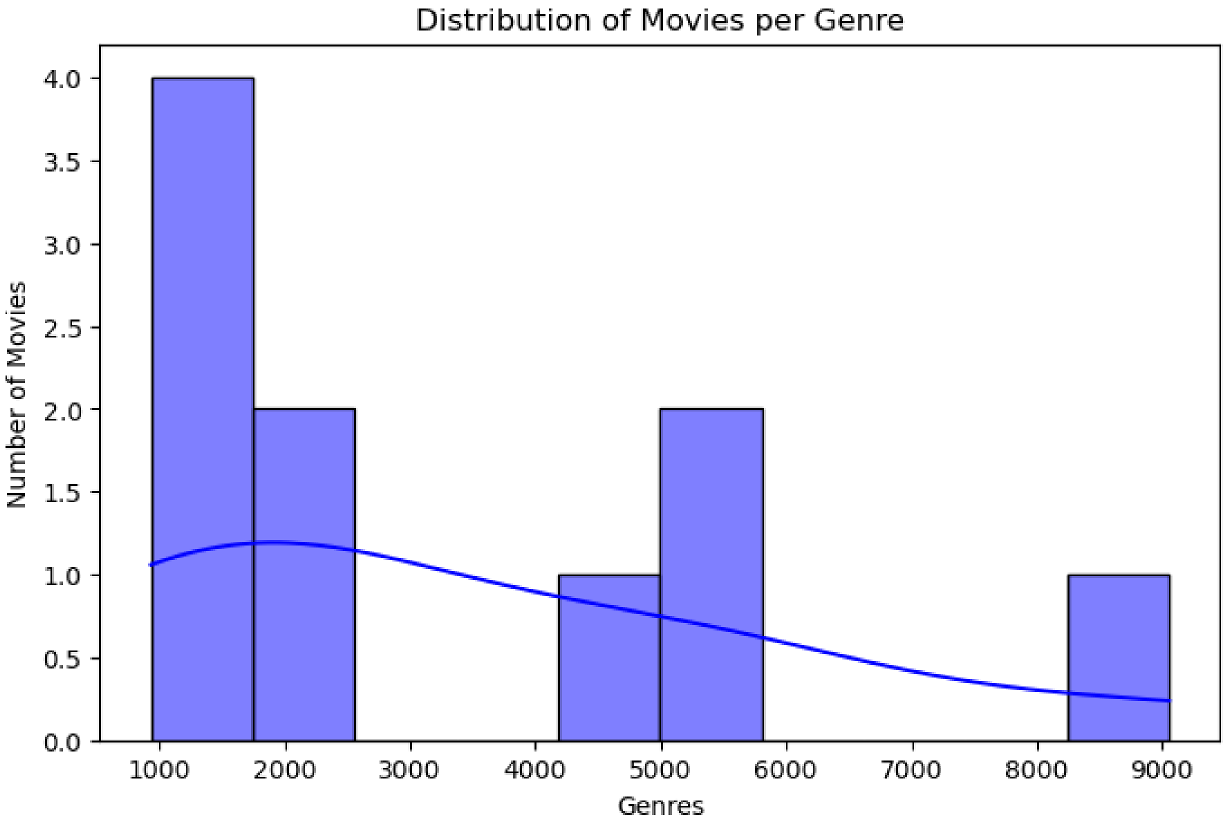
```
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=top_10_genres.values, y=top_10_genres.index,
palette="coolwarm")
```



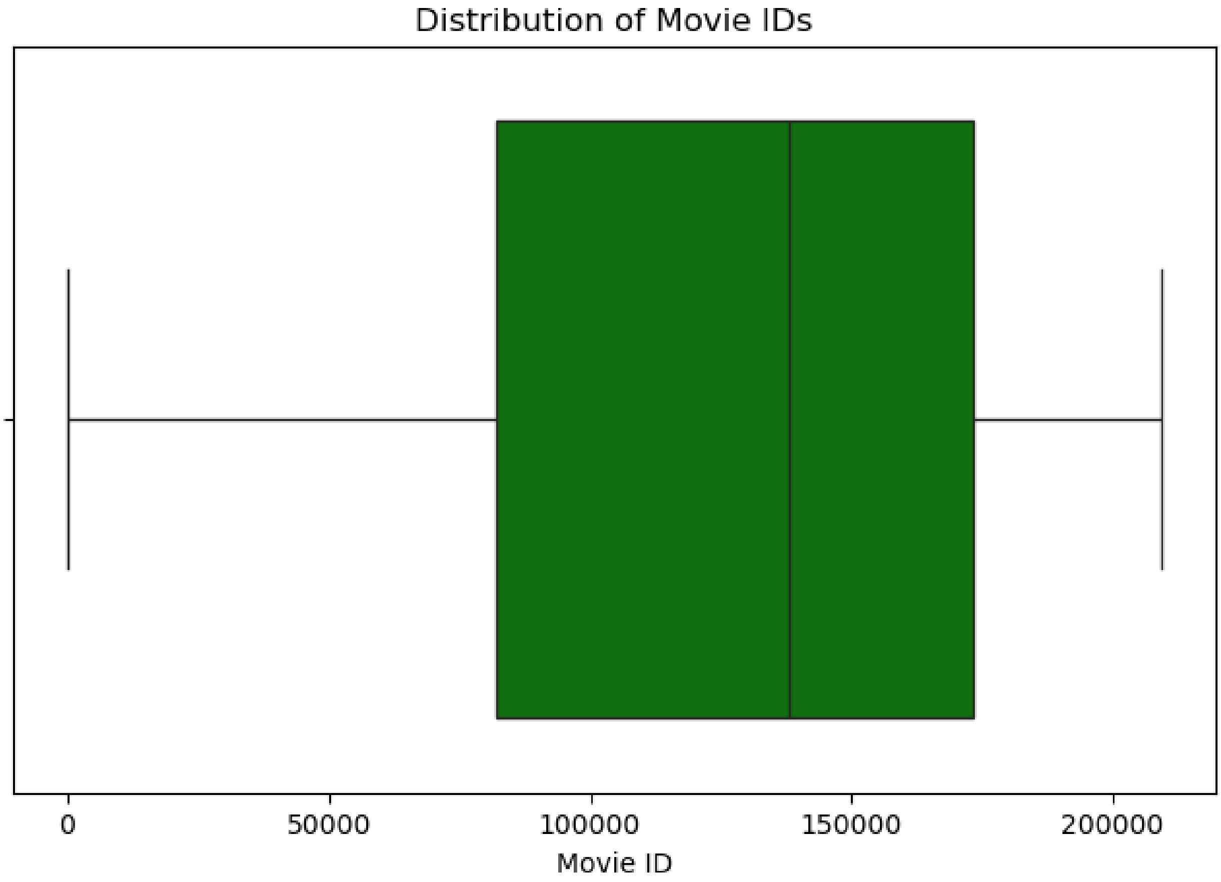
```
# Count occurrences of each genre
genre_counts = mov['movie_type'].value_counts().head(10)  # Top 10
genres

# Create a Pie Chart
plt.figure(figsize=(8, 8))
plt.pie(genre_counts, labels=genre_counts.index, autopct='%1.1f%%',
startangle=140, colors=plt.cm.Paired.colors)
plt.title("Top 10 Movie Genres Distribution")
plt.show()
```



```
plt.figure(figsize=(8, 5))
sns.histplot(genre_counts, bins=10, kde=True, color="blue")
plt.xlabel("Genres")
plt.ylabel("Number of Movies")
plt.title("Distribution of Movies per Genre")
plt.show()
```



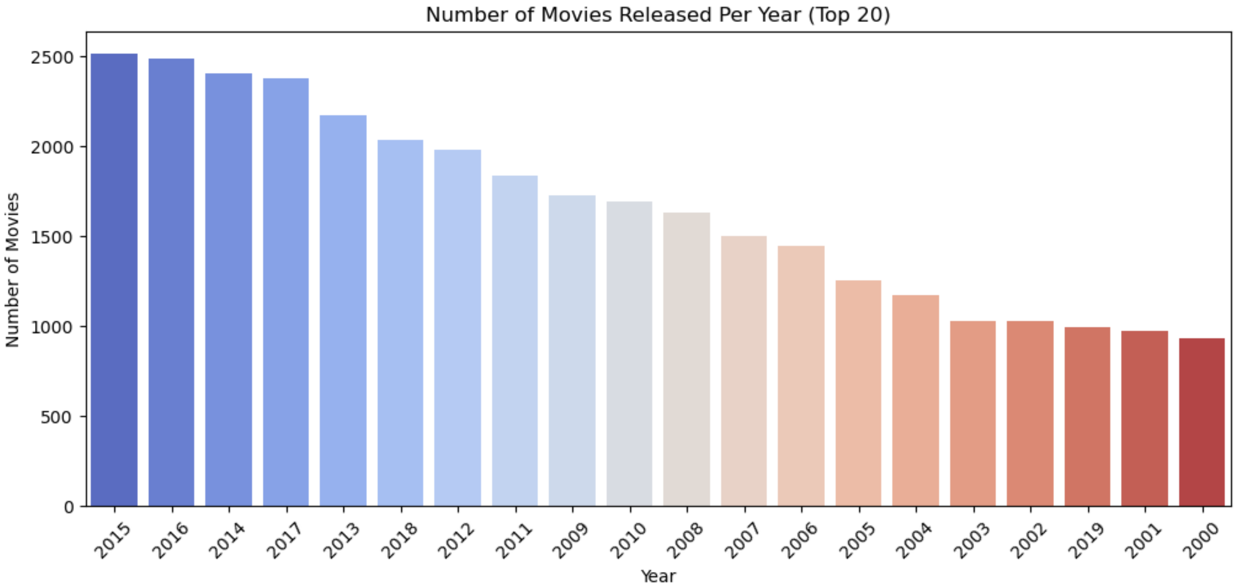
```
plt.figure(figsize=(8, 5))
sns.boxplot(x=mov["movie_id"], color="green")
plt.xlabel("Movie ID")
plt.title("Distribution of Movie IDs")
plt.show()
```



```
# Extract Year from Title (if present in dataset)
mov["year"] = mov["movie_name"].str.extract(r"\((\d{4})\)")

# Drop missing years
mov = mov.dropna(subset=["year"])

# Convert to integer
mov["year"] = mov["year"].astype(int)

# Plot Movies Per Year
plt.figure(figsize=(12, 5))
sns.countplot(x=mov["year"], palette="coolwarm",
order=mov["year"].value_counts().index[:20])
plt.xticks(rotation=45)
plt.xlabel("Year")
plt.ylabel("Number of Movies")
plt.title("Number of Movies Released Per Year (Top 20)")
plt.show()
```

```
C:\Users\Mi\AppData\Local\Temp\ipykernel_13872\1753450607.py: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-versus-a-copy
  mov["year"] = mov["movie_name"].str.extract(r"\((\d{4})\)")
C:\Users\Mi\AppData\Local\Temp\ipykernel_13872\1753450607.py:12:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x=mov["year"], palette="coolwarm",
order=mov["year"].value_counts().index[:20])
```



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Display first few rows
print(mov.head())
```

```
   movie_id                          movie_name  \
0         1                    Toy Story (1995)
1         2                      Jumanji (1995)
```

```
2         3             Grumpier Old Men (1995)
3         4            Waiting to Exhale (1995)
4         5  Father of the Bride Part II (1995)

                                    movie_type  year
0  Adventure|Animation|Children|Comedy|Fantasy  1995
1                   Adventure|Children|Fantasy  1995
2                               Comedy|Romance  1995
3                         Comedy|Drama|Romance  1995
4                                       Comedy  1995
```

```
# Display first few rows
print(mov.columns)
```

```
Index(['movie_id', 'movie_name', 'movie_type', 'year'],
dtype='object')
```

```
# Split dataset into train and test sets
trainset, testset = train_test_split(mov, test_size=0.2,
random_state=42)

print(mov.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 62013 entries, 0 to 62422
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   movie_id    62013 non-null  int64 
 1   movie_name  62013 non-null  object
 2   movie_type  62013 non-null  object
 3   year        62013 non-null  int32 
dtypes: int32(1), int64(1), object(2)
memory usage: 2.1+ MB
None
```

```
print(mov.dropna())
```

```
       movie_id                          movie_name  \
0             1                    Toy Story (1995)   
1             2                      Jumanji (1995)   
2             3             Grumpier Old Men (1995)   
3             4            Waiting to Exhale (1995)   
4             5  Father of the Bride Part II (1995)   
...         ...                                 ...   
62418    209157                           We (2018)   
62419    209159           Window of the Soul (2001)   
62420    209163                    Bad Poems (2018)   
62421    209169                 A Girl Thing (2001)   
62422    209171      Women of Devil's Island (1962)   
```

```
                                        movie_type  year
0      Adventure|Animation|Children|Comedy|Fantasy  1995
1                       Adventure|Children|Fantasy  1995
2                                   Comedy|Romance  1995
3                             Comedy|Drama|Romance  1995
4                                           Comedy  1995
...                                            ...   ...
62418                                        Drama  2018
62419                                  Documentary  2001
62420                                 Comedy|Drama  2018
62421                           (no genres listed)  2001
62422                       Action|Adventure|Drama  1962

[62013 rows x 4 columns]
```

```
target_column = mov.columns[-1]  # Assuming the last column is the target
print(f'Target column identified: {target_column}')
```

```
Target column identified: year
```

```
label_encoders = {}
for column in mov.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    mov[column] = le.fit_transform(mov[column])
    label_encoders[column] = le

X = mov.drop(columns=[target_column])
y = mov[target_column]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5)  # Default is 5 neighbors
model.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
y_pred = model.predict(X_test)

#Calculate Mean Squared Error (MSE)
mse_value = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error (MSE): {mse_value:.2f}')

#Calculate Root Mean Squared Error (RMSE)
rmse_value = np.sqrt(mse_value)
```

```
print(f'Root Mean Squared Error (RMSE): {rmse_value:.2f}')

#Calculate Precision and Recall
precision = precision_score(y_test, y_pred, average='weighted',
zero_division=1)
recall = recall_score(y_test, y_pred, average='weighted',
zero_division=1)

print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
```

```
Mean Squared Error (MSE): 1576.31
Root Mean Squared Error (RMSE): 39.70
Precision: 0.09
Recall: 0.07
```