```
!pip install pymongo --quiet
!pip install python-dotenv --quiet

from pymongo import MongoClient
import pandas as pd
from pprint import pprint
import matplotlib.pyplot as plt
from IPython.display import display
import seaborn as sns
from collections import Counter
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/1.4 MB ? eta -:--:--
━━━━━━━━╺━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.3/1.4 MB 8.3 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━  1.4/1.4 MB 22.7 MB/s
eta 0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 17.2
MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/313.6 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 14.8 MB/s eta
0:00:00
```

## Connection to MongoDB and get Documents

```
# Configuration
MONGO_URI = ""
DATABASE_NAME = ""
COLLECTION_NAME = ""

def connect_to_mongodb(mongo_uri):
    """Function to connect to mongodb client.
    Return mongodb client if successfull."""

    # Connect to server
    mongo_client = MongoClient(mongo_uri)

    # Ping to server
    try:
      mongo_client.admin.command('ping')
      print("Pinged your deployment. You successfully connected to
MongoDB!")
    except Exception as e:
      raise Exception(f"Connection failed: {e}")

    return mongo_client

def get_database(mongo_client, database_name):
    """Function to get database from mongodb client."
    Return database if successfull."""
```

```python
    # Get database
    try:
      database = mongo_client[database_name]
      print(f"Database '{database_name}' connected successfully!")
    except Exception as e:
      raise Exception(f"Connection failed: {e}")

    return database

def get_collection(database, collection_name):
    """Function to get collection from database."
    Return collection if successfull."""

    # Get collection
    try:
      collection = database[collection_name]
      print(f"Collection '{collection_name}' connected successfully!")
    except Exception as e:
      raise Exception(f"Connection failed: {e}")

    return collection

def get_all_documents(collection):
    """Function to get all documents from collection.
        Return list of documents.
    """
    all_documents = []
    try:
        documents = collection.find()
        for document in documents:
          all_documents.append(document)
    except Exception as e:
      raise Exception(f"Connection failed: {e}")

    return all_documents

# Connect to MongoDB and get collection
mongo_client = connect_to_mongodb(MONGO_URI)
database = get_database(mongo_client, DATABASE_NAME)
collection = get_collection(database, COLLECTION_NAME)
```

```
Pinged your deployment. You successfully connected to MongoDB!
Database 'Film' connected successfully!
Collection 'word_embedding_preprocessed' connected successfully!
```

```python
# Get all documents
all_documents = get_all_documents(collection)

# Print some samples
all_documents[:1]
```

```
[{'_id': ObjectId('6839ee0bdba46ba351845660'),
  'id': 'tt0006621',
  'cleaned_description': ['when',
   'isabel',
   'carlisle',
   'mistakenly',
   'belief',
   'that',
   'her',
   'husband',
   'richard',
   'love',
   'barbara',
   'hare',
   'she',
   'leaf',
   'him',
   'their',
   'two',
   'child',
   'she',
   'doe',
   'nothing',
   'to',
   'correct',
   'report',
   'that',
   'she',
   'ha',
   'been',
   'killed',
   'in',
   'train',
   'wreck',
   'so',
   'richard',
   'believing',
   'himself',
   'to',
   'be',
   'widower',
   'marries',
   'barbara',
   'after',
   'few',
   'month',
   'isabel',
   'longs',
   'to',
   'see',
   'her',
   'child',
   'so',
   'disguising',
   'herself',
   'get',
   'job',
   'a',
   'their',
   'governess',
   'then',
   'when',
   'her',
   'son',
   'becomes',
   'ill',
   'call',
   'out',
   'for',
   'his',
   'mother',
   'isabel',
   'throw',
   'off',
   'her',
   'disguise',
   'go',
   'to',
   'comfort',
   'him',
   'he',
   'dy',
   'in',
   'her',
   'arm',
   'discovering',
   'isabel',
   'with',
   'boy',
   'richard',
   'immediately',
   'forgives',
   'her',
   'for',
   'having',
   'left',
   'him',
   'child',
   'isabel',
   'can',
   'not',
   'forgive',
   'herself',
   'soon',
   'dy',
   'of',
   'grief'],
  'metadata': {'film_name': 'East Lynne',
   'image_link':
'https://m.media-amazon.com/images/M/MV5BM2E3MDgzNjItZjMzYi00ZWZiLTgwM
WMtNWM2ZDAwOTg0MzBhXkEyXkFqcGc@.jpg',
   'is_adult': 0,
   'start_year': 1916,
   'runtime_minutes': 50,
   'genres': 'Drama',
   'rating': 5.5,
   'votes': 51,
   'directors': 'Bertram Bracken',
   'writers': 'Mary Elizabeth Braddon, Mary Murillo, Mrs. Henry
Wood'},
  'original_description': 'When Isabel Carlisle mistakenly believes
that her husband Richard loves Barbara Hare, she leaves him and their
two children. She does nothing to correct the report that she has been
killed in a train wreck, and so Richard, believing himself to be a
widower, marries Barbara. After a few months, Isabel longs to see her
children and so, disguising herself, gets a job as their governess.
Then, when her son becomes ill and calls out for his mother, Isabel
throws off her disguise and goes to comfort him, but he dies in her
arms. Discovering Isabel with the boy, Richard immediately forgives
her for having left him and the children, but Isabel cannot forgive
herself, and soon dies of grief.'}]
```

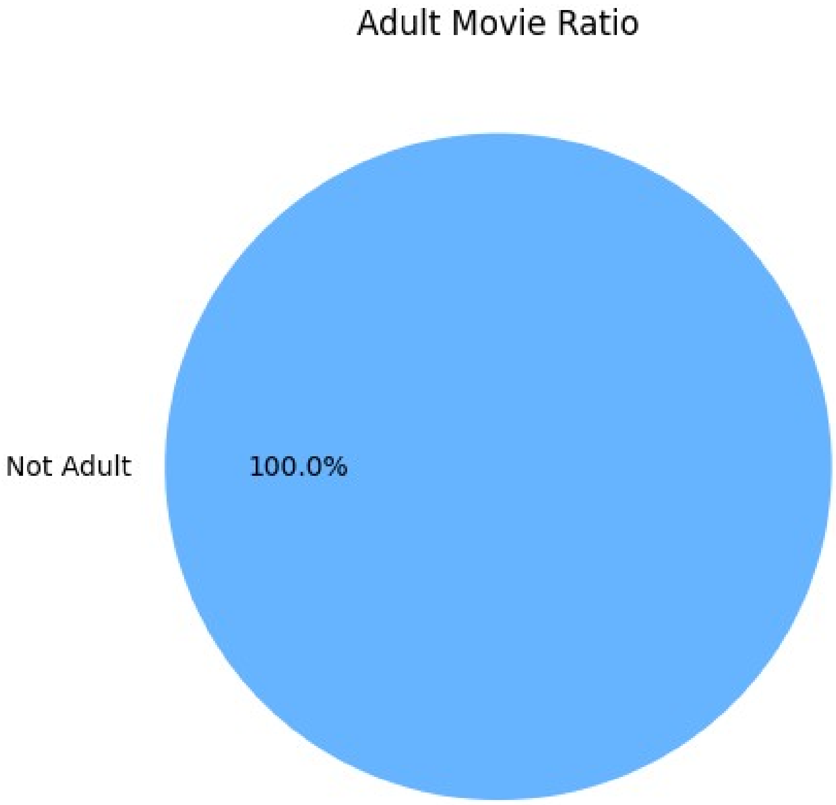
```python
df = pd.DataFrame(all_documents)

# Mở rộng metadata
meta_df = pd.json_normalize(df["metadata"])
df = df.join(meta_df).drop(columns=["metadata"])
df = df.drop(columns=['_id', 'id'])

print("Total Movies:", len(df))
print(df.info())

print("\nMissing information:")
print(df.isnull().sum())
```

```
Total Movies: 9504
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9504 entries, 0 to 9503
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
```

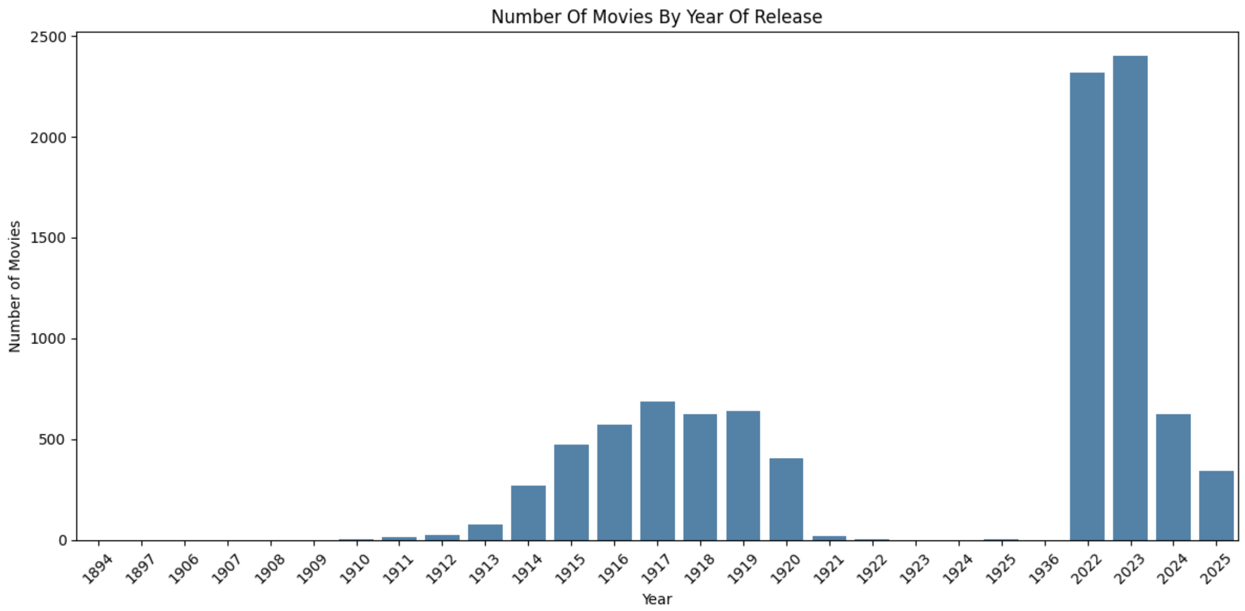
```
---  ------                --------------  -----
 0   cleaned_description   9504 non-null   object
 1   original_description  9504 non-null   object
 2   film_name             9504 non-null   object
 3   image_link            9504 non-null   object
 4   is_adult              9504 non-null   int64
 5   start_year            9504 non-null   int64
 6   runtime_minutes       7668 non-null   float64
 7   genres                9365 non-null   object
 8   rating                6017 non-null   float64
 9   votes                 6017 non-null   float64
 10  directors             9402 non-null   object
 11  writers               8972 non-null   object
dtypes: float64(3), int64(2), object(7)
memory usage: 891.1+ KB
None

Missing information:
cleaned_description       0
original_description      0
film_name                 0
image_link                0
is_adult                  0
start_year                0
runtime_minutes        1836
genres                  139
rating                 3487
votes                  3487
directors               102
writers                 532
dtype: int64
```

## Movie Information Analysis

```
#Tỷ lệ phim người lớn / không người lớn
plt.figure(figsize=(5,5))
df["is_adult"].value_counts().plot.pie(autopct="%.1f%%", labels=["Not
Adult", "Adult"], colors=["#66b3ff", "#ff9999"])
plt.title("Adult Movie Ratio")
plt.ylabel("")
plt.tight_layout()
plt.show()
```



```python
#Năm phát hành
plt.figure(figsize=(12, 6))

df["start_year"] = pd.to_numeric(df["start_year"], errors="coerce")
df_filtered = df.dropna(subset=["start_year"])
df_filtered["start_year"] = df_filtered["start_year"].astype(int)

sns.countplot(data=df_filtered, x="start_year", color='steelblue')
plt.xticks(rotation=45)
plt.title("Number Of Movies By Year Of Release")
plt.xlabel("Year")
plt.ylabel("Number of Movies")
plt.tight_layout()
plt.show()
```



```
#Phân tích thời lượng phim
runtime = df["runtime_minutes"].dropna()

# Tính IQR để loại các giá trị outlier
Q1 = runtime.quantile(0.25)
Q3 = runtime.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df_runtime = runtime[(runtime >= lower_bound) & (runtime <=
upper_bound)]
df_runtime.describe()
```

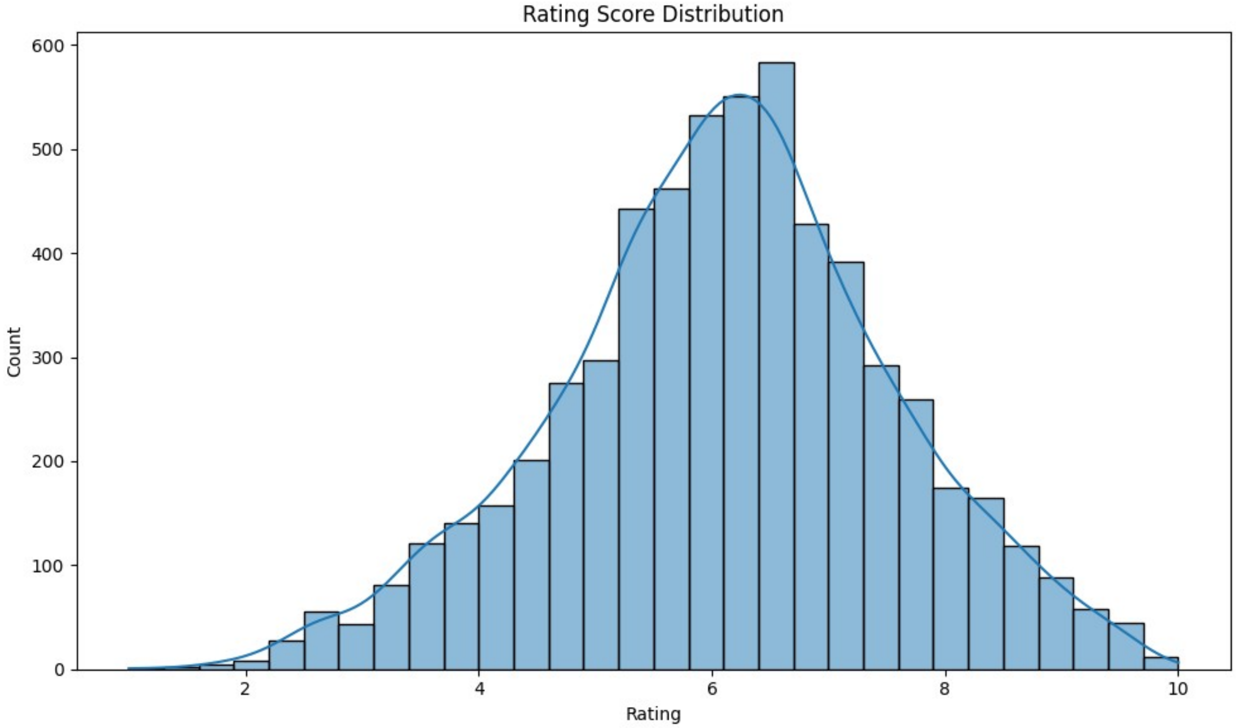
```
count    7599.000000
mean       82.543098
std        28.523298
min         3.000000
25%        50.000000
50%        84.000000
75%       101.000000
max       177.000000
Name: runtime_minutes, dtype: float64
```

```
#Rating
rating_series = df["rating"].dropna()
rating_series.describe()
```

```
count    6017.000000
mean        6.090128
std         1.440450
```

```
min          1.000000
25%          5.200000
50%          6.100000
75%          7.000000
max         10.000000
Name: rating, dtype: float64
```

```
plt.figure(figsize=(10, 6))
sns.histplot(rating_series, bins=30, kde=True)
plt.title("Rating Score Distribution")
plt.xlabel("Rating")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```



```
# Lâ´y top 10 phim rating cao nhâ´t và thâ´p nhâ´t
rating_high = df.nlargest(10, "rating")[["film_name", "directors",
"start_year", "runtime_minutes", "votes",
"rating"]].reset_index(drop=True)
rating_low = df.nsmallest(10, "rating")[["film_name", "directors",
"start_year", "runtime_minutes", "votes",
"rating"]].reset_index(drop=True)

print("Top 10 Highest Rated Movies:")
display(rating_high)
```

```
print("\nTop 10 Lowest Rated Movies:")
display(rating_low)
```

```
Top 10 Highest Rated Movies:
```

```
{"summary":"{\n  \"name\": \"rating_high\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"film_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"Tears
of Blood\",\n          \"The Place in Between\",\n          \"Baldy
for the Blind\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"directors\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Joshua Clay\",\n          \"Laura Perez\",\n
\"Drea Castro\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_year\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 2022,\n
\"max\": 2025,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2024,\n          2023,\n          2022\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"runtime_minutes\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
22.169799277395363,\n        \"min\": 78.0,\n        \"max\": 139.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          139.0,\n
103.0,\n          91.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"votes\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 24.959745369071552,\n        \"min\": 7.0,\n        \"max\":
78.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
9.0,\n          8.0,\n          54.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.09660917830792998,\n        \"min\": 9.7,\n        \"max\": 10.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          10.0,\n
9.8,\n          9.7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"rating_high"}
```

```
Top 10 Lowest Rated Movies:
```

```
{"summary":"{\n  \"name\": \"rating_low\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"film_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"A
Man's Fight\",\n          \"Love Song and Power\",\n          \"One
Hour\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"directors\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Thomas N. Heffron\",\n          \"Erik Krefeld, Eddel
Martinez\",\n          \"Edwin L. Hollywood, Paul McAllister\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"start_year\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
54,\n        \"min\": 1915,\n        \"max\": 2024,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          1920,\n
2024,\n          2023\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"runtime_minutes\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 23.101870672554387,\n        \"min\":
50.0,\n        \"max\": 125.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          90.0,\n          89.0,\n          50.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"votes\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
427.7260416045143,\n        \"min\": 8.0,\n        \"max\": 1096.0,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          28.0,\n
8.0,\n          43.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.2836272984824353,\n        \"min\": 1.0,\n        \"max\":
1.9,\n        \"num_unique_values\": 6,\n        \"samples\": [\n
1.0,\n          1.4,\n          1.9\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"rating_low"}
```

```python
# Lấy top 10 phim có votes cao nhất và thấp nhất
votes_high = df.nlargest(10, "votes")[["film_name", "directors",
"start_year", "runtime_minutes", "rating",
"votes"]].reset_index(drop=True)
votes_low = df.nsmallest(10, "votes")[["film_name", "directors",
"start_year", "runtime_minutes", "rating",
"votes"]].reset_index(drop=True)

print("Top 10 Highest Votes Movies:")
display(votes_high)

print("\nTop 10 Lowest Votes Movies:")
display(votes_low)
```

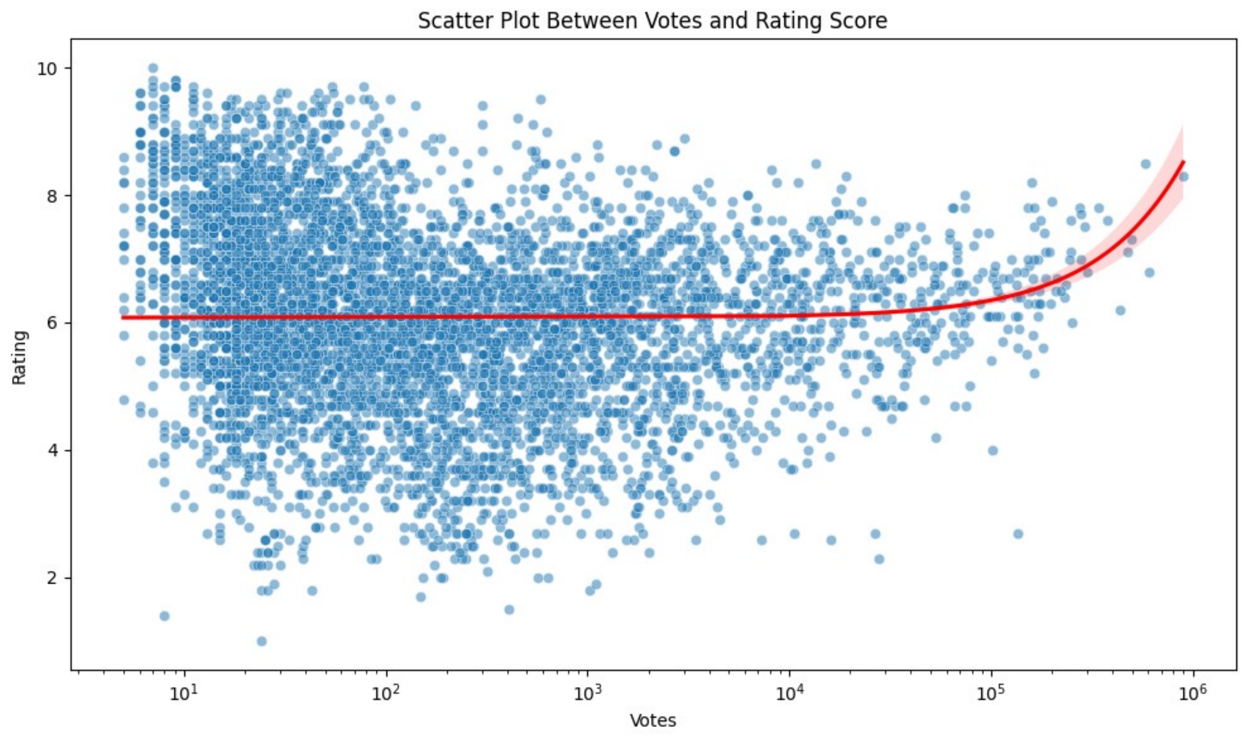
```
Top 10 Highest Votes Movies:
```

```
{"summary":"{\n  \"name\": \"votes_high\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"film_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Nope\",\n          \"Barbie\",\n          \"Thor: Love and
Thunder\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"directors\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Jordan Peele\",\n          \"Greta Gerwig\",\n
\"Taika Waititi\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_year\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 2022,\n
\"max\": 2024,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          2023,\n          2022,\n          2024\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"runtime_minutes\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
22.867735640708577,\n        \"min\": 114.0,\n        \"max\": 180.0,\
n        \"num_unique_values\": 10,\n        \"samples\": [\n
130.0,\n          114.0,\n          118.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"rating\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.7062420107709382,\n        \"min\": 6.2,\n        \"max\": 8.5,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          7.8,\n
6.8,\n          6.2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"votes\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 182137.90032841722,\n        \"min\": 288503.0,\n
\"max\": 894728.0,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          299655.0,\n          609292.0,\n
434464.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"votes_high"}
```

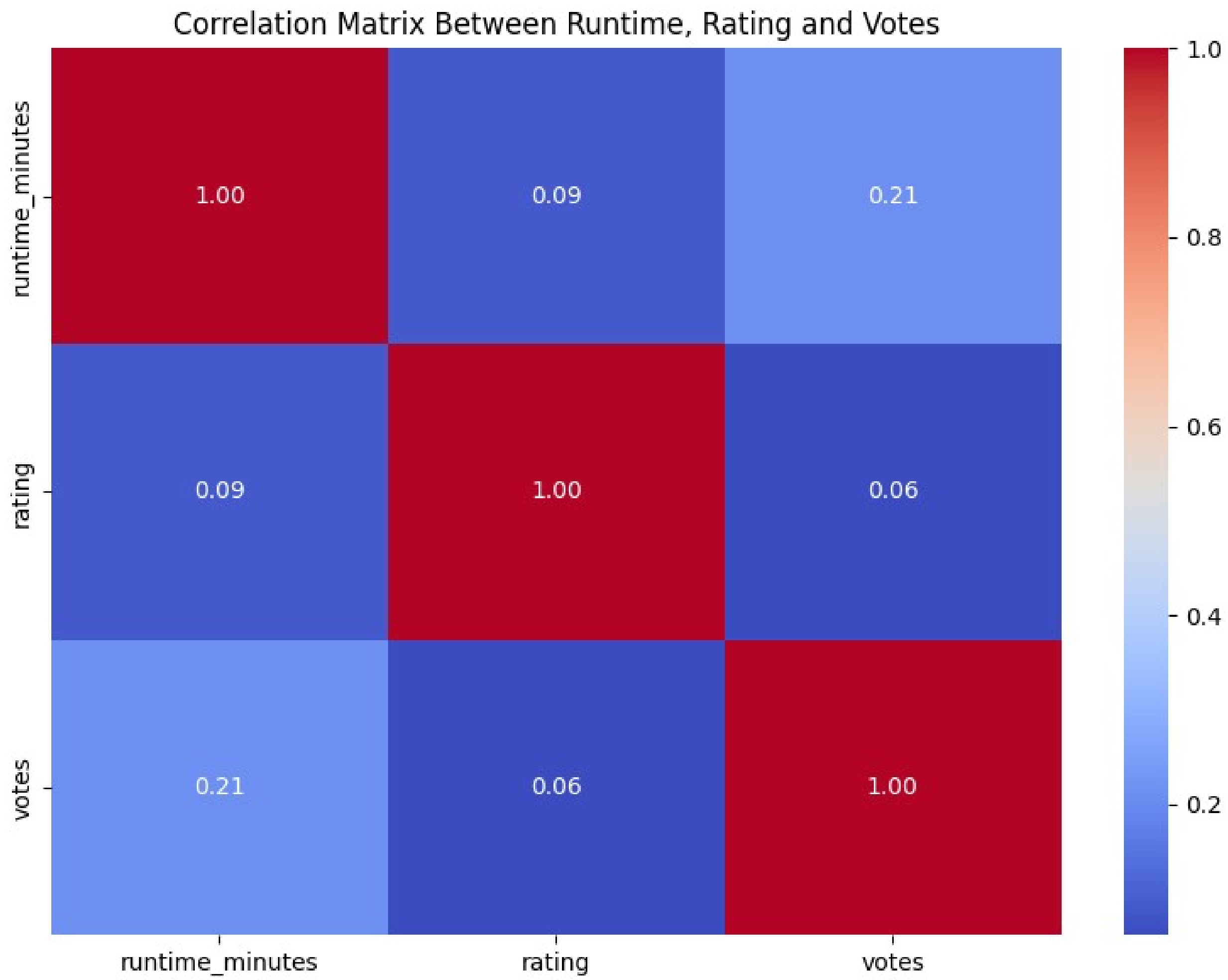
```
Top 10 Lowest Votes Movies:
```

```
{"summary":"{\n  \"name\": \"votes_low\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"film_name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Apocalypse Love\",\n          \"Manuela\",\n          \"A Sunken
Place\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"directors\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Vera VanGuard\",\n          \"Clara Cullen\",\n
\"Ronan O'Leary\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_year\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 2022,\n
\"max\": 2024,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          2022,\n          2023,\n          2024\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"runtime_minutes\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
19.27347054027029,\n        \"min\": 65.0,\n        \"max\": 115.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          78.0,\n
90.0,\n          115.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rating\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1.2479316220584096,\n        \"min\": 4.8,\n        \"max\":
8.6,\n        \"num_unique_values\": 10,\n        \"samples\": [\n
6.4,\n          7.4,\n          7.8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"votes\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0.0,\n
\"min\": 5.0,\n        \"max\": 5.0,\n        \"num_unique_values\":
1,\n        \"samples\": [\n          5.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"votes_low"}
```

```python
#Scatterplot votes và rating
df_votes_rating = df[["votes", "rating"]].dropna()

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_votes_rating, x="votes", y="rating",
alpha=0.5)
sns.regplot(data=df_votes_rating, x="votes", y="rating",
scatter=False, color='red')
plt.title("Scatter Plot Between Votes and Rating Score")
plt.xlabel("Votes")
plt.ylabel("Rating")
plt.xscale('log')
plt.tight_layout()
plt.show()
```

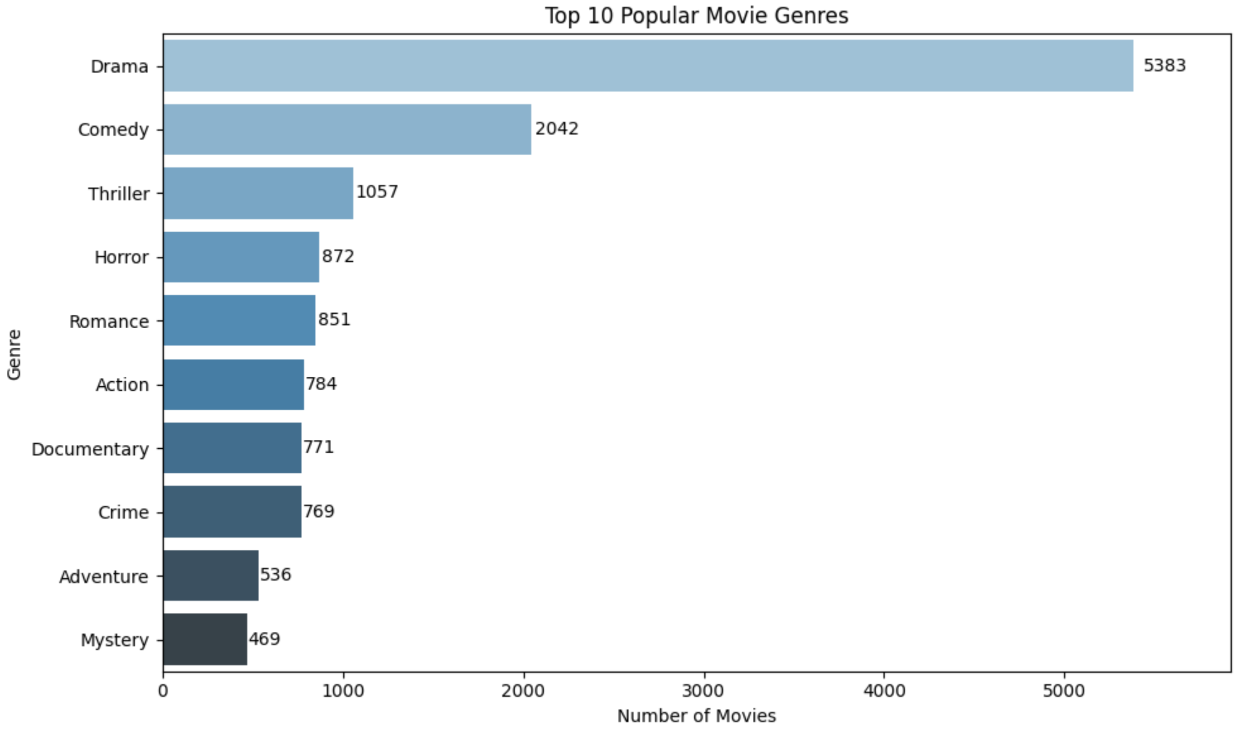


```
#Heatmap
df_corr = df[df["runtime_minutes"].between(lower_bound, upper_bound)]
[["runtime_minutes", "rating", "votes"]].dropna()

plt.figure(figsize=(8, 6))
sns.heatmap(df_corr.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix Between Runtime, Rating and Votes")
plt.tight_layout()
plt.show()
```

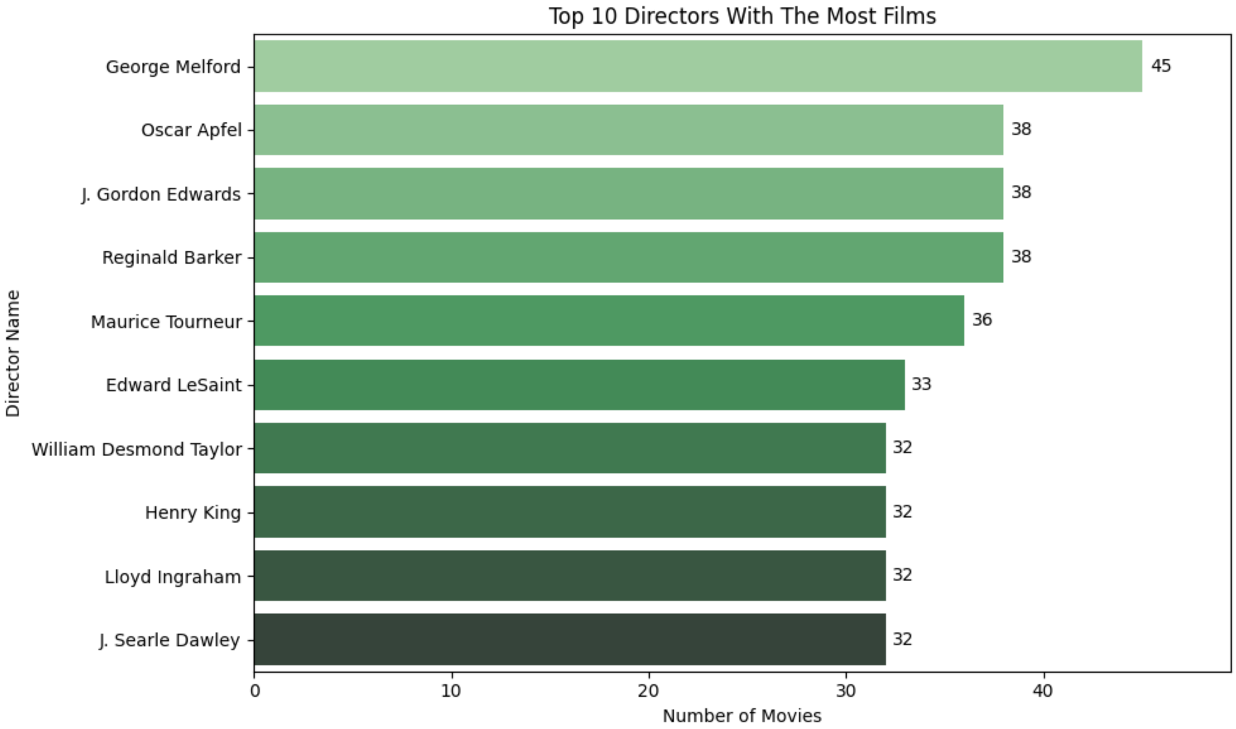


```python
#In thể loại và top 10 phổ biến
genre_series = df["genres"].dropna().str.split(",").explode()
unique_genres = sorted(genre_series.str.strip().unique())
print(f"Number of Movie Genres: {len(unique_genres)} genres")
print("\nList of Movie Genres:")
print(", ".join(unique_genres))

top_genres = genre_series.value_counts().head(10)
top_genres_df = top_genres.reset_index()
top_genres_df.columns = ["genre", "count"]

plt.figure(figsize=(10, 6))
ax = sns.barplot(x="count", y="genre", hue="genre",
data=top_genres_df, palette="Blues_d", legend=False)

for i, v in enumerate(top_genres_df["count"]):
    ax.text(v * 1.01, i, str(v), va="center")

plt.xlim(0, top_genres_df["count"].max() * 1.1)
plt.title("Top 10 Popular Movie Genres")
plt.xlabel("Number of Movies")
plt.ylabel("Genre")
plt.tight_layout()
plt.show()
```

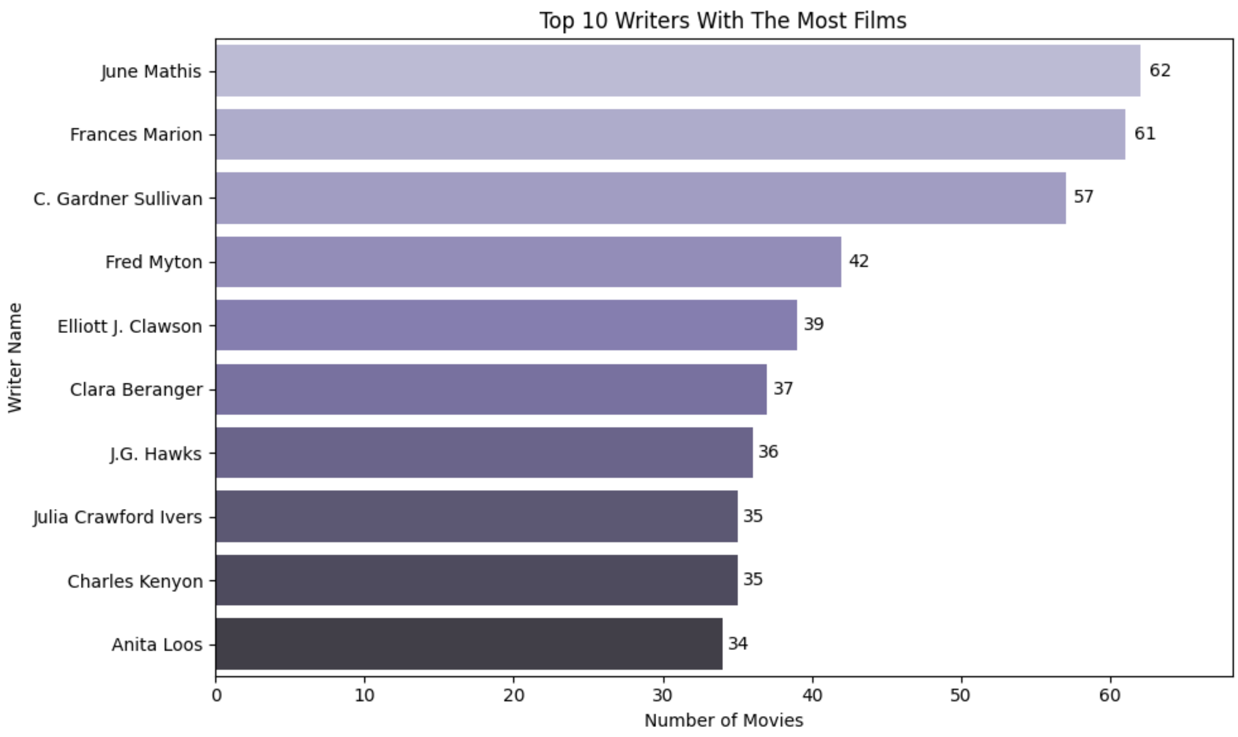
```
Number of Movie Genres: 24 genres

List of Movie Genres:
Action, Adventure, Animation, Biography, Comedy, Crime, Documentary,
Drama, Family, Fantasy, History, Horror, Music, Musical, Mystery,
News, Reality-TV, Romance, Sci-Fi, Sport, Talk-Show, Thriller, War,
Western
```



```
#Top 10 đạo diê˜n
directors = df["directors"].dropna().str.split(",").explode()
top_directors = directors.value_counts().head(10)
top_directors_df = top_directors.reset_index()
top_directors_df.columns = ["director", "count"]

plt.figure(figsize=(10, 6))
ax = sns.barplot(x="count", y="director", hue="director",
data=top_directors_df, palette="Greens_d", legend=False)

for i, v in enumerate(top_directors_df["count"]):
    ax.text(v * 1.01, i, str(v), va="center")

plt.xlim(0, top_directors_df["count"].max() * 1.1)
plt.title("Top 10 Directors With The Most Films")
plt.xlabel("Number of Movies")
plt.ylabel("Director Name")
plt.tight_layout()
plt.show()
```



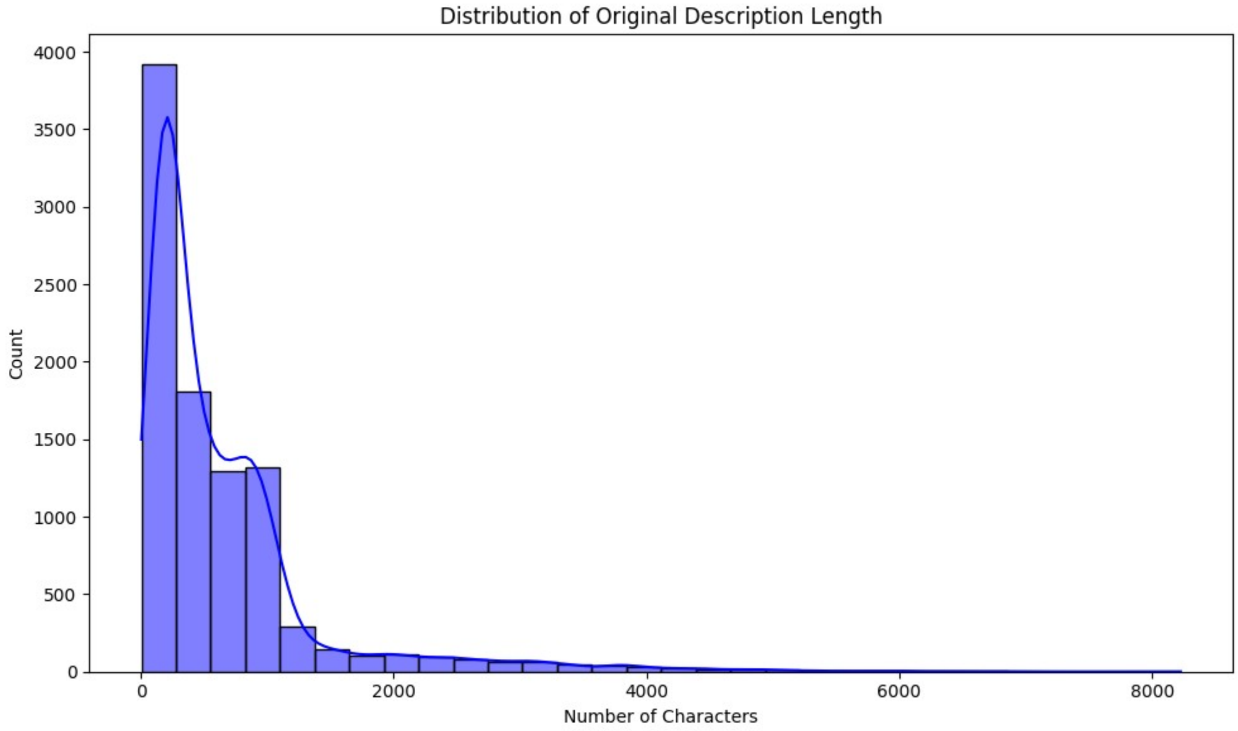
```
#Top 10 biên kịch
writer_series =
df["writers"].dropna().str.split(",").explode().str.strip()
top_writers = writer_series.value_counts().head(10)
top_writers_df = top_writers.reset_index()
top_writers_df.columns = ["writer", "count"]

plt.figure(figsize=(10, 6))
ax = sns.barplot(x="count", y="writer", hue="writer",
data=top_writers_df, palette="Purples_d", legend=False)

for i, v in enumerate(top_writers_df["count"]):
    ax.text(v * 1.01, i, str(v), va="center")

plt.xlim(0, top_writers_df["count"].max() * 1.1)
plt.title("Top 10 Writers With The Most Films")
plt.xlabel("Number of Movies")
plt.ylabel("Writer Name")
plt.tight_layout()
plt.show()
```



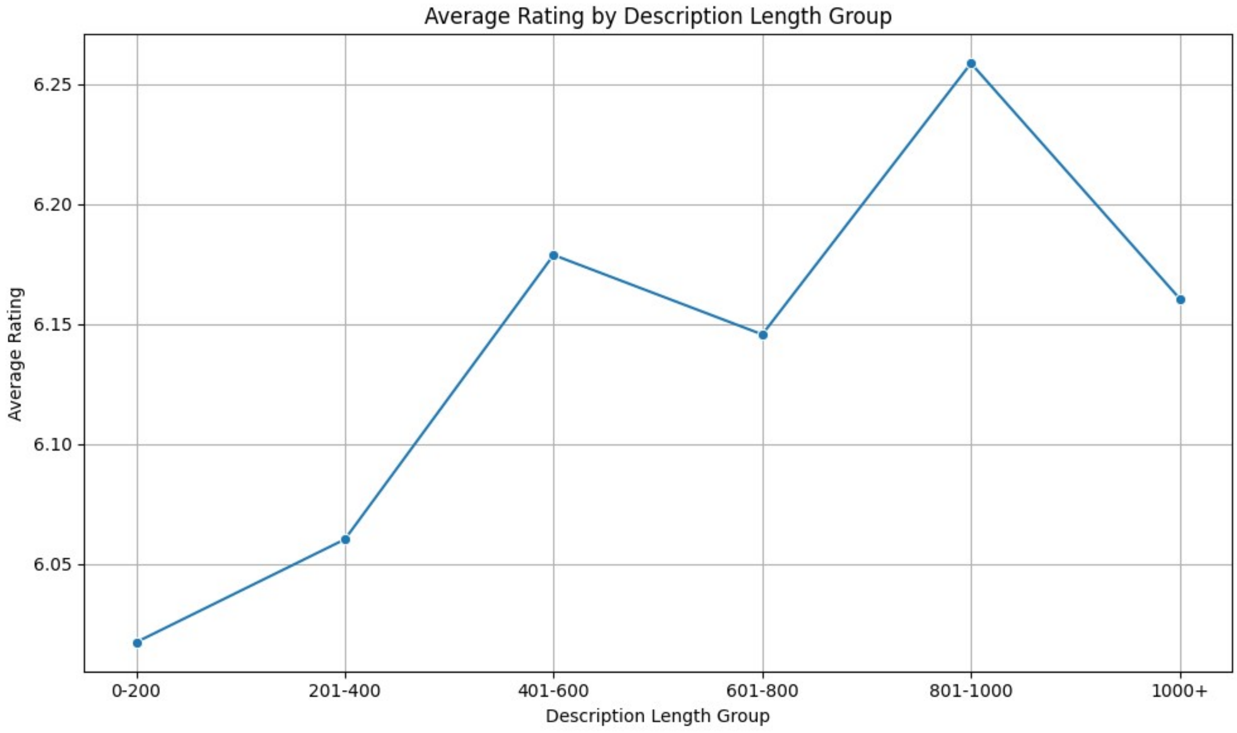
# Word Frequency Analysis

```
#Phân tích độ dài mô tả
df["description_length"] =
df["original_description"].astype(str).apply(len)

print("Movie description length statistics:")
print(df["description_length"].describe())

plt.figure(figsize=(10, 6))
sns.histplot(df["description_length"], bins=30, kde=True,
color="blue")
plt.title("Distribution of Original Description Length")
plt.xlabel("Number of Characters")
plt.tight_layout()
plt.show()
```

```
Movie description length statistics:
count    9504.000000
mean      657.577020
std       789.409125
min         4.000000
25%       198.000000
50%       384.500000
75%       847.000000
max      8227.000000
Name: description_length, dtype: float64
```



```
#So sánh giữa độ dài mô tả và rating
df["desc_length_bin"] = pd.cut(df["description_length"], bins=[0, 200,
400, 600, 800, 1000, 2000],
                               labels=["0-200", "201-400", "401-600",
"601-800", "801-1000", "1000+"])
grouped = df.groupby("desc_length_bin", observed=True)
["rating"].mean().dropna()

plt.figure(figsize=(10, 6))
sns.lineplot(x=grouped.index.astype(str), y=grouped.values,
marker="o")
plt.title("Average Rating by Description Length Group")
plt.xlabel("Description Length Group")
plt.ylabel("Average Rating")
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
#Đếm tổng từ, tổng từ khác nhau
all_words = [word for desc in df["cleaned_description"] if
isinstance(desc, list) for word in desc]

total_words = len(all_words)
print(f"Total Words: {total_words:,}")

# Số từ khác nhau
diff_words = set(all_words)
print(f"Total Different Words: {len(diff_words):,}")

Total Words: 911,049
Total Different Words: 36,622

word_counts = Counter(all_words)

#Số từ xuất hiện chỉ 1 lần
word_once = [word for word, count in word_counts.items() if count ==
1]
print(f"Number of Words Appearing Only Once: {len(word_once):,}")

Number of Words Appearing Only Once: 13,974

#Top 20 từ phổ biến
top_20 = pd.DataFrame(word_counts.most_common(20), columns=["Word",
"Frequency"])
top_20["Percent"] = (top_20["Frequency"] / total_words * 100).round(2)
```

```
print("\nTop 20 Common Words:\n")
print(top_20)
```

```

Top 20 Common Words:

     Word  Frequency  Percent
0      to      38302     4.20
1      of      28978     3.18
2      in      20453     2.24
3     her      18944     2.08
4     his      18576     2.04
5      he      13656     1.50
6    with      10605     1.16
7    that      10223     1.12
8     she       9338     1.02
9     for       8682     0.95
10     by       7399     0.81
11    him       7160     0.79
12    who       6592     0.72
13     on       6181     0.68
14      a       5981     0.66
15     ha       5822     0.64
16   from       5371     0.59
17   when       5264     0.58
18     at       4619     0.51
19  their       4489     0.49
```

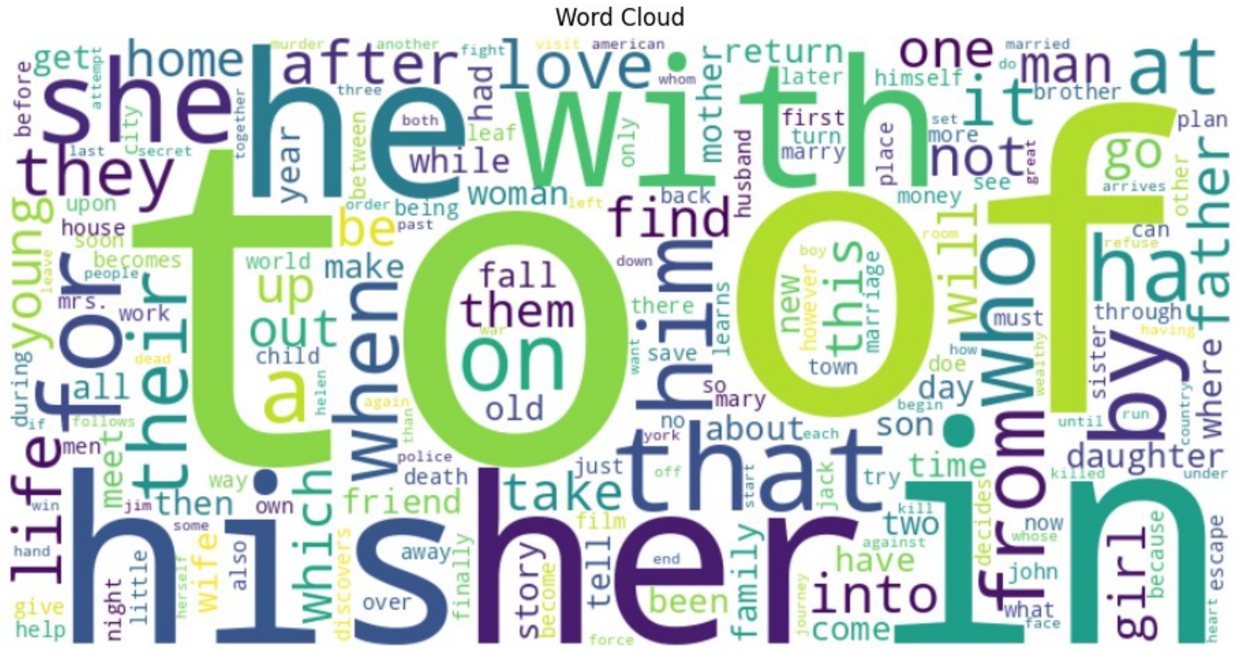
```
#Bigram
df["cleaned_description"] = df["cleaned_description"].apply(lambda x:
" ".join(x) if isinstance(x, list) else str(x))

vectorizer = CountVectorizer(ngram_range=(2, 2), min_df=5)
X = vectorizer.fit_transform(df["cleaned_description"])

bigrams = vectorizer.get_feature_names_out()
sum_bigrams = X.sum(axis=0).A1
bigram_freq = pd.Series(sum_bigrams,
index=bigrams).sort_values(ascending=False)

bigram_df = bigram_freq.head(20).reset_index()
bigram_df.columns = ["Bigram", "Frequency"]

print("\nTop 20 Most Frequent Bigrams:\n")
print(bigram_df.to_string(index=False))
```

```

Top 20 Most Frequent Bigrams:

    Bigram  Frequency
    of his       2113
    of her       1501
   that he       1475
   in love       1437
     to be       1347
    to his       1167
    to her       1118
 love with       1117
  with her       1102
  that she       1094
  his wife       1073
   ha been       1024
    in his       1008
her father       1004
    her to       1001
    him to       1001
     go to        977
     he ha        863
   fall in        847
    one of        846
```

```python
#Trigram
vectorizer = CountVectorizer(ngram_range=(3, 3), min_df=5)
X = vectorizer.fit_transform(df["cleaned_description"])

trigrams = vectorizer.get_feature_names_out()
sum_trigrams = X.sum(axis=0).A1
trigram_freq = pd.Series(sum_trigrams,
index=trigrams).sort_values(ascending=False)

trigram_df = trigram_freq.head(20).reset_index()
trigram_df.columns = ["Trigram", "Frequency"]

print("\nTop 20 Most Frequent Trigrams:\n")
print(trigram_df.to_string(index=False))
```

```

Top 20 Most Frequent Trigrams:

       Trigram  Frequency
  in love with       1102
  fall in love        809
   in order to        350
   to new york        205
    that he ha        203
 love with her        187
   in new york        186
   who ha been        173
  that he will        147
   that she ha        147
    in time to        141
  to marry him        140
  in search of        135
      to go to        128
  with help of        118
   take her to        113
      he go to        110
    he doe not       108
of her husband       102
   to save her        102
```

```python
wordcloud = WordCloud(width=800, height=400,
background_color="white").generate_from_frequencies(word_counts)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Word Cloud")
plt.tight_layout()
plt.show()
```



Word Cloud

```python
#  Đê´m từ xuâ´t hiện nhiê`u hơn 0.1% tổng sô´ từ
threshold = 0.001 * total_words
common_words = [(word, count) for word, count in word_counts.items()
if count >= threshold]
print(f"\nNumber of words appearing > 0.1% of total words
({int(threshold)} times): {len(common_words)}")

#  Đê´m từ xuâ´t hiện nhiê`u hơn 0.01% tổng sô´ từ
threshold = 0.0001 * total_words
```

```
common_words = [(word, count) for word, count in word_counts.items()
if count >= threshold]
print(f"\nNumber of words appearing > 0.01% of total words
({int(threshold)} times): {len(common_words)}")
```

```

Number of words appearing > 0.1% of total words (911 times): 105

Number of words appearing > 0.01% of total words (91 times): 1266
```