Language Dictionary

In [ ]:
```
# dictionary for ISO_639_1 languages column

"""
We need this dictionary for data cleaning purposes. In
the "Spoken Languages" Column in the 5000 movies file for
example, we can we how some entries include taglines in the
language, therefore we can use this dictionary for lookup
instead for complete values
"""

iso_639_1_languages = {
    "aa": "Afar",
    "ab": "Abkhazian",
    "ae": "Avestan",
    "af": "Afrikaans",
    "ak": "Akan",
    "am": "Amharic",
    "an": "Aragonese",
    "ar": "Arabic",
    "as": "Assamese",
    "av": "Avaric",
    "ay": "Aymara",
    "az": "Azerbaijani",
    "ba": "Bashkir",
    "be": "Belarusian",
    "bg": "Bulgarian",
    "bh": "Bihari languages",
    "bi": "Bislama",
    "bm": "Bambara",
    "bn": "Bengali",
    "bo": "Tibetan",
    "br": "Breton",
    "bs": "Bosnian",
    "ca": "Catalan; Valencian",
    "ce": "Chechen",
    "ch": "Chamorro",
```

```
    "cn": "Chinese",
    "co": "Corsican",
    "cr": "Cree",
    "cs": "Czech",
    "cu": "Church Slavic; Old Slavonic; Church Slavonic; Old Bulgarian; Old Church Slavonic",
    "cv": "Chuvash",
    "cy": "Welsh",
    "da": "Danish",
    "de": "German",
    "dv": "Divehi; Dhivehi; Maldivian",
    "dz": "Dzongkha",
    "ee": "Ewe",
    "el": "Greek, Modern (1453-)",
    "en": "English",
    "eo": "Esperanto",
    "es": "Spanish; Castilian",
    "et": "Estonian",
    "eu": "Basque",
    "fa": "Persian",
    "ff": "Fulah",
    "fi": "Finnish",
    "fj": "Fijian",
    "fo": "Faroese",
    "fr": "French",
    "fy": "Western Frisian",
    "ga": "Irish",
    "gd": "Gaelic; Scottish Gaelic",
    "gl": "Galician",
    "gn": "Guarani",
    "gu": "Gujarati",
    "gv": "Manx",
    "ha": "Hausa",
    "he": "Hebrew",
    "hi": "Hindi",
    "ho": "Hiri Motu",
    "hr": "Croatian",
    "ht": "Haitian; Haitian Creole",
    "hu": "Hungarian",
    "hy": "Armenian",
    "hz": "Herero",
    "ia": "Interlingua (International Auxiliary Language Association)",
    "id": "Indonesian",
```

```
    "ie": "Interlingue; Occidental",
    "ig": "Igbo",
    "ii": "Sichuan Yi; Nuosu",
    "ik": "Inupiaq",
    "io": "Ido",
    "is": "Icelandic",
    "it": "Italian",
    "iu": "Inuktitut",
    "ja": "Japanese",
    "jv": "Javanese",
    "ka": "Georgian",
    "kg": "Kongo",
    "ki": "Kikuyu; Gikuyu",
    "kj": "Kuanyama; Kwanyama",
    "kk": "Kazakh",
    "kl": "Kalaallisut; Greenlandic",
    "km": "Central Khmer",
    "kn": "Kannada",
    "ko": "Korean",
    "kr": "Kanuri",
    "ks": "Kashmiri",
    "ku": "Kurdish",
    "kv": "Komi",
    "kw": "Cornish",
    "ky": "Kirghiz; Kyrgyz",
    "la": "Latin",
    "lb": "Luxembourgish; Letzeburgesch",
    "lg": "Ganda",
    "li": "Limburgan; Limburger; Limburgish",
    "ln": "Lingala",
    "lo": "Lao",
    "lt": "Lithuanian",
    "lu": "Luba-Katanga",
    "lv": "Latvian",
    "mg": "Malagasy",
    "mh": "Marshallese",
    "mi": "Maori",
    "mk": "Macedonian",
    "ml": "Malayalam",
    "mn": "Mongolian",
    "mr": "Marathi",
    "ms": "Malay",
```

```
    "mt": "Maltese",
    "my": "Burmese",
    "na": "Nauru",
    "nb": "Norwegian Bokmål",
    "nd": "North Ndebele",
    "ne": "Nepali",
    "ng": "Ndonga",
    "nl": "Dutch; Flemish",
    "nn": "Norwegian Nynorsk",
    "no": "Norwegian",
    "nr": "South Ndebele",
    "nv": "Navajo; Navaho",
    "ny": "Chichewa; Chewa; Nyanja",
    "oc": "Occitan (post 1500)",
    "oj": "Ojibwa",
    "om": "Oromo",
    "or": "Oriya",
    "os": "Ossetian; Ossetic",
    "pa": "Panjabi; Punjabi",
    "pi": "Pali",
    "pl": "Polish",
    "ps": "Pushto; Pashto",
    "pt": "Portuguese",
    "qu": "Quechua",
    "rm": "Romansh",
    "rn": "Rundi",
    "ro": "Romanian; Moldavian; Moldovan",
    "ru": "Russian",
    "rw": "Kinyarwanda",
    "sa": "Sanskrit",
    "sc": "Sardinian",
    "sd": "Sindhi",
    "se": "Northern Sami",
    "sg": "Sango",
    "sh": "Serbo-Croatian",
    "si": "Sinhala; Sinhalese",
    "sk": "Slovak",
    "sl": "Slovenian",
    "sm": "Samoan",
    "sn": "Shona",
    "so": "Somali",
    "sq": "Albanian",
```

```
    "sr": "Serbian",
    "ss": "Swati",
    "st": "Sotho, Southern",
    "su": "Sundanese",
    "sv": "Swedish",
    "sw": "Swahili",
    "ta": "Tamil",
    "te": "Telugu",
    "tg": "Tajik",
    "th": "Thai",
    "ti": "Tigrinya",
    "tk": "Turkmen",
    "tl": "Tagalog",
    "tn": "Tswana",
    "to": "Tonga (Tonga Islands)",
    "tr": "Turkish",
    "ts": "Tsonga",
    "tt": "Tatar",
    "tw": "Twi",
    "ty": "Tahitian",
    "ug": "Uighur; Uyghur",
    "uk": "Ukrainian",
    "ur": "Urdu",
    "uz": "Uzbek",
    "ve": "Venda",
    "vi": "Vietnamese",
    "vo": "Volapük",
    "wa": "Walloon",
    "wo": "Wolof",
    "xh": "Xhosa",
    "yi": "Yiddish",
    "yo": "Yoruba",
    "za": "Zhuang; Chuang",
    "zh": "Chinese",
    "zu": "Zulu"
}
```

Import Libraries

In [ ]:
```
from concurrent.futures import ThreadPoolExecutor, as_completed
from sklearn.preprocessing import MultiLabelBinarizer
import requests
from bs4 import BeautifulSoup
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
import ast
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, MultiLabelBinarizer
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_percentage_error
from sklearn.model_selection import KFold
import ast, csv
```

In [ ]:
```
movies_df = pd.read_csv("tmdb_5000_movies.csv")
print(movies_df.dtypes)
```

```
budget                  int64
genres                 object
homepage               object
id                      int64
keywords               object
original_language      object
original_title         object
overview               object
popularity            float64
production_companies   object
production_countries   object
release_date           object
revenue                 int64
runtime               float64
spoken_languages       object
status                 object
tagline                object
title                  object
vote_average          float64
vote_count              int64
dtype: object
```

Data Cleaning Setup

Part 1:

In this part of the Data Cleaning, we try to identify the movies in the dataset which has been misidentified as having a revenue of 0. For example, the movie "Chiamatemi Francesco - Il Papa della gente" has a revenue of 0 in the dataset but in actuality when we look it up it has a revenue of around 3.91 million dollars. In order to accomodate for this, we make use of the OMDb API as well as Web Scraping the IMDB website where we first identify all the movies which have a revenue of 0; then we send each title into the OMDB API to get the IMDB url for that movie, from which we can then perform web scraping to find the global revenue for the movie on the IMDB website - if it is available. The API has a free limit of 1000 daily uses and there were over 1400 instances of 0 revenue movies, so we used two API keys and concurrency to make this process faster. Once we find the revenue for the movie on the IMDB website, we update that movie's value in the dataframe - while also looking for other NaN values that could be present such as runtime, release date, and overview.

In [ ]:
```
def get_gross_worldwide(imdb_url):
  headers = {"User-Agent": "Mozilla/5.0"}
  response = requests.get(imdb_url, headers=headers)
  soup = BeautifulSoup(response.text, "html.parser")

  # Find the span in whic gross revenue is located to be extracted

  gross_label_span = soup.find("span", string="Gross worldwide")
  if gross_label_span:
    container = gross_label_span.find_parent("li")
    if container:
      spans = container.find_all("span")
      for span in spans:
        if span != gross_label_span:
          return span.text.strip()
  return None

def get_movie_details(title, api_key):

  # calling OMDb API using given API Key and movie title

  base_url = "http://www.omdbapi.com/"
  params = {"t": title, "apikey": api_key}
  response = requests.get(base_url, params=params)
  data = response.json()
  return data if data.get("Response") == "True" else None

def process_movie(title_and_key):
  title, api_key = title_and_key
  result = {'title': title}

  data = get_movie_details(title, api_key)
  if not data:
    print(f"Error, could not find data for movie: {title}")
    return result

  # If we data comes back, we see if parameters were valid

  result['released'] = data.get('Released')
  runtime_str = data.get('Runtime')
  result['runtime'] = None
```

```
  if runtime_str and runtime_str != 'N/A':

    # if we have the runtime, convert it to float and insert to df

    try:
      result['runtime'] = float(runtime_str.split()[0])
    except ValueError:
      print(f"Could not parse runtime for {title}: {runtime_str}")
  else:
    print(f"Runtime not available for movie: {title}")

  result['overview'] = data.get('Plot')

  imdb_id = data.get('imdbID')
  if imdb_id:
    url = f"https://www.imdb.com/title/{imdb_id}/"

    # once we extract IMDB url from API Response, get revenue from
    # Web Scrape

    revenue = get_gross_worldwide(url)
    if revenue:
      revenue_clean = revenue.replace('$', '').replace(',', '')
      try:
        result['revenue'] = int(revenue_clean)
      except ValueError:
        print(f"Could not parse revenue for {title}: {revenue}")
        result['revenue'] = None
    else:
      print(f"Unable to scrape Revenue for movie: {title}")
  else:
    print(f"No IMDb ID for movie: {title}")
    result['revenue'] = None

  return result


# defining API Keys
api_key_1 = "eab3a590"
api_key_2 = "ae1fa689"

# Extracting titles of the movies with revenue of 0
```

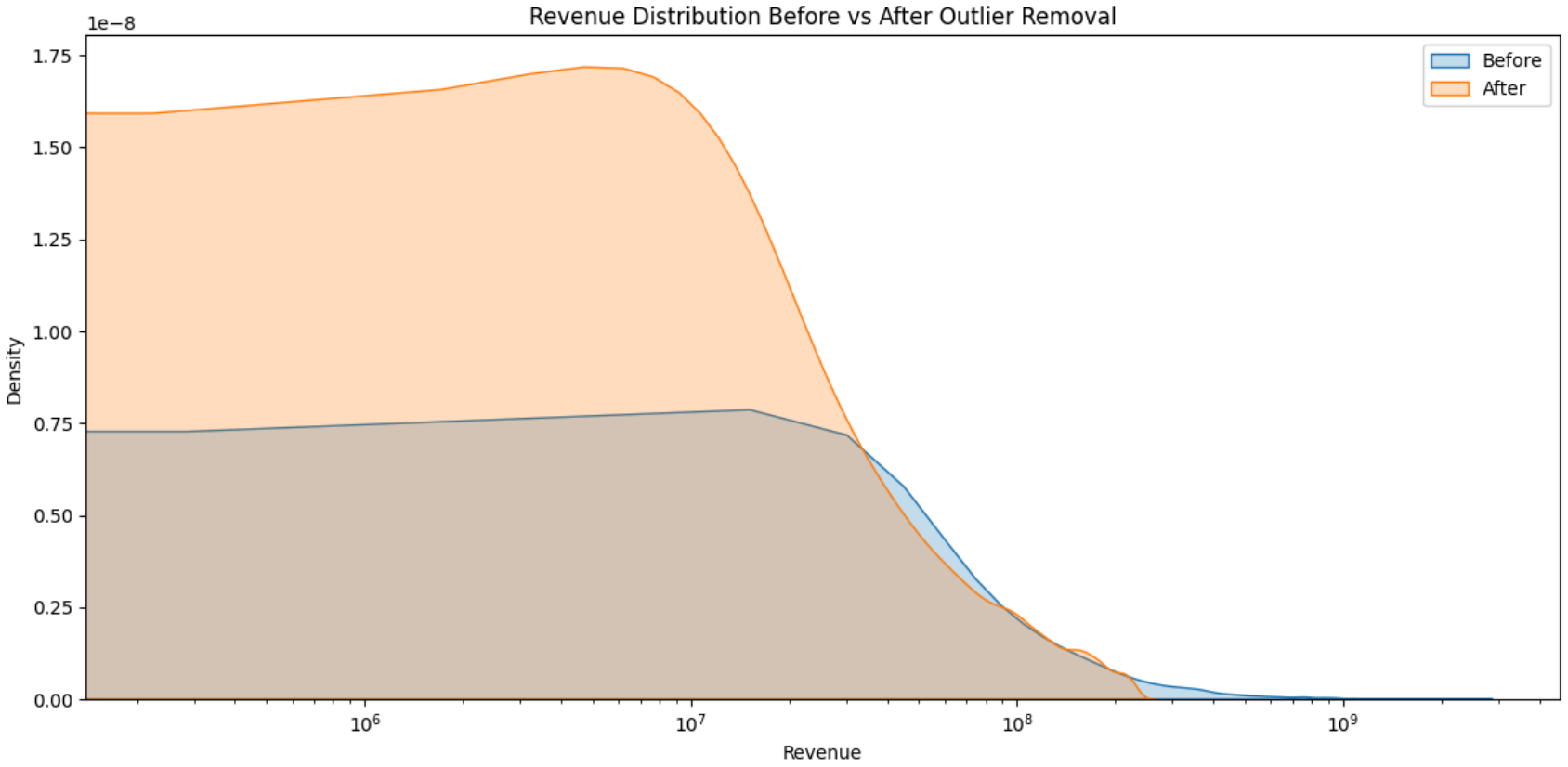
```
movie_titles = movies_df[movies_df['revenue'] == 0]['title']

# Split work between both API Keys
titles_keys = [(title, api_key_1 if i < 712 else api_key_2) for i, title in enumerate(movie_titles)]

results = []
with ThreadPoolExecutor(max_workers=10) as executor:
  futures = [executor.submit(process_movie, tk) for tk in titles_keys]
  for i, future in enumerate(as_completed(futures), start=1):
    result = future.result()
    results.append(result)
    if i % 100 == 0:
      print(f"Processed {i} movies...")


# updating Dataframes after threads have completed
for result in results:
  title = result['title']
  mask = movies_df["title"] == title

  # if the value in the df is NaN, and we have information from the API then we fill it
  # into the dataframe for the movie
  if pd.isna(movies_df.loc[mask, 'release_date']).any() and result.get('released'):
      movies_df.loc[mask, 'release_date'] = result['released']

  if pd.isna(movies_df.loc[mask, 'runtime']).any() and result.get('runtime') is not None:
      movies_df.loc[mask, 'runtime'] = result['runtime']

  if pd.isna(movies_df.loc[mask, 'overview']).any() and result.get('overview'):
      movies_df.loc[mask, 'overview'] = result['overview']

  if result.get('revenue') is not None:
      movies_df.loc[mask, 'revenue'] = result['revenue']

# coerce the release_date column to datetime structure
movies_df['release_date'] = pd.to_datetime(movies_df['release_date'], errors='coerce')
movies_df['release_date'] = movies_df['release_date'].dt.strftime('%Y-%m-%d')

print("Writing data to file...")
movies_df.to_csv('part1_movies.csv', index=False)

print("Finished processing for all movies")
```

Removing Outliers and Creating Density graph

In [ ]:
```
filtered_df = pd.read_csv('part1_movies.csv')

print(filtered_df.isnull().sum().to_string())
print("\n")

"""
After printing the number of NaN values in the DataFrame per column,
we can see that the column with the most NaN values was 'homepage' with
3091 and the second was 'tagline' with 844. Since these are just string
values we can replace them with 'N/A' as a string
"""


filtered_df['homepage'] = filtered_df['homepage'].fillna('N/A')
filtered_df['tagline'] = filtered_df['tagline'].fillna('N/A')

revenue = filtered_df['revenue']
Q1, Q3 = revenue.quantile(0.25), revenue.quantile(0.75)
IQR = Q3 - Q1

lower, upper = Q1 - 1.5 * IQR, Q3 + 1.5 * IQR

clean_df = filtered_df[(revenue >= lower) & (revenue <= upper)].copy()
print(f"Before removing outliers {len(filtered_df)} rows ")
print(f"After removing outliers {len(clean_df)} rows")
plt.figure(figsize=(12, 6))

# Updated to use fill=True instead of shade=True
sns.kdeplot(filtered_df['revenue'], label='Before', fill=True)
sns.kdeplot(clean_df['revenue'], label='After', fill=True)

plt.xscale('log')  # Keep log scale for better distribution visibility
plt.title('Revenue Distribution Before vs After Outlier Removal')
plt.xlabel('Revenue')
plt.ylabel('Density')
plt.legend()
```

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

```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   1
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    1
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0


Before removing outliers 4803 rows
After removing outliers 4335 rows
```

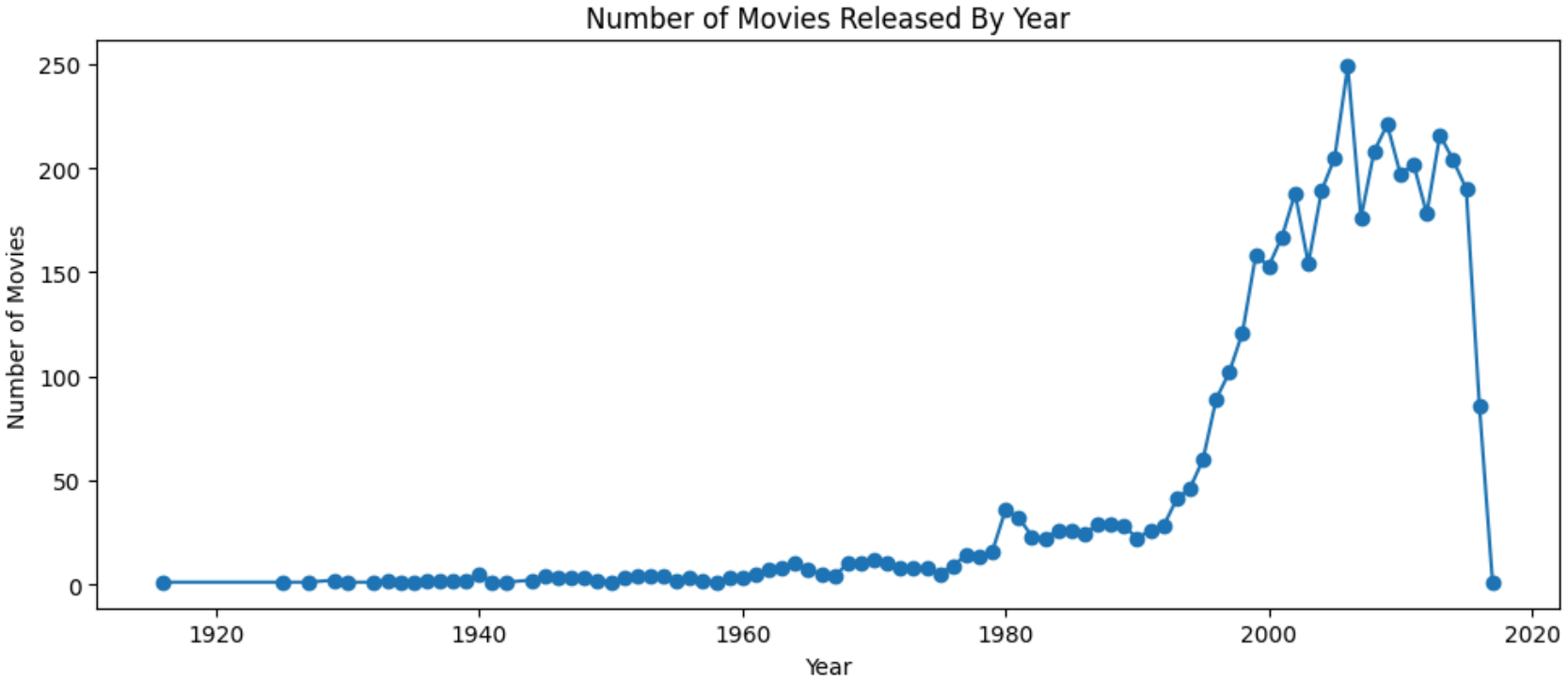


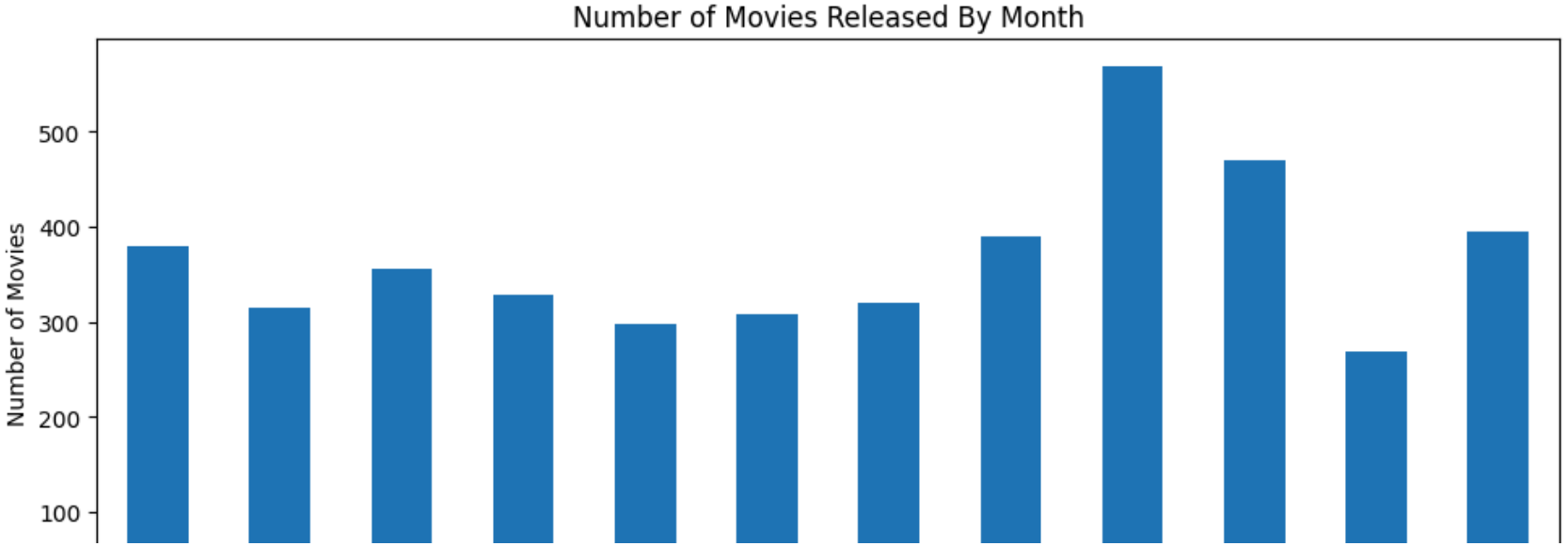
In [ ]:
```
"""
Cleaning the remaining NaN Values in the Data Frame
"""

runtime_title = clean_df[clean_df['runtime'].isna()]['title']
clean_df.loc[clean_df['title'] == runtime_title.iloc[0], 'runtime'] = float(81)

release_title = clean_df[clean_df['release_date'].isna()]['title']
clean_df.loc[clean_df['title'] == release_title.iloc[0], 'release_date'] = "09/09/2015"

overview_title = clean_df[clean_df['overview'].isna()]['title']
clean_df.loc[clean_df['title'] == overview_title.iloc[0], 'overview'] = "The life of Frank Sinatra, as an actor and s
```

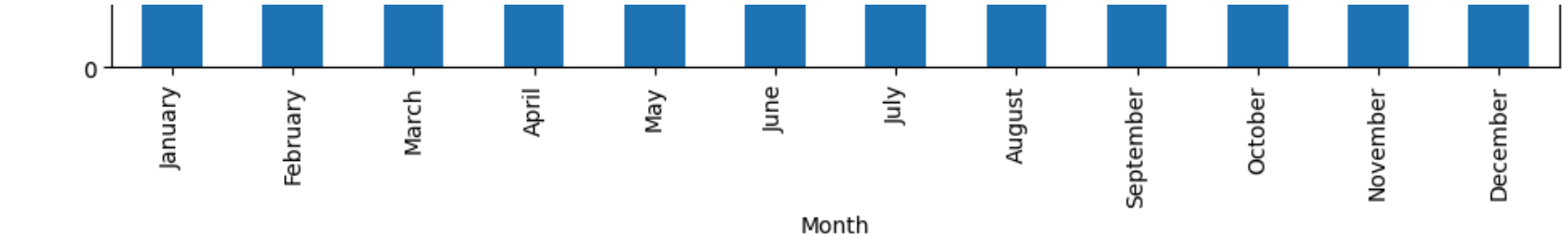
Counting movies by year, month, and day of the week

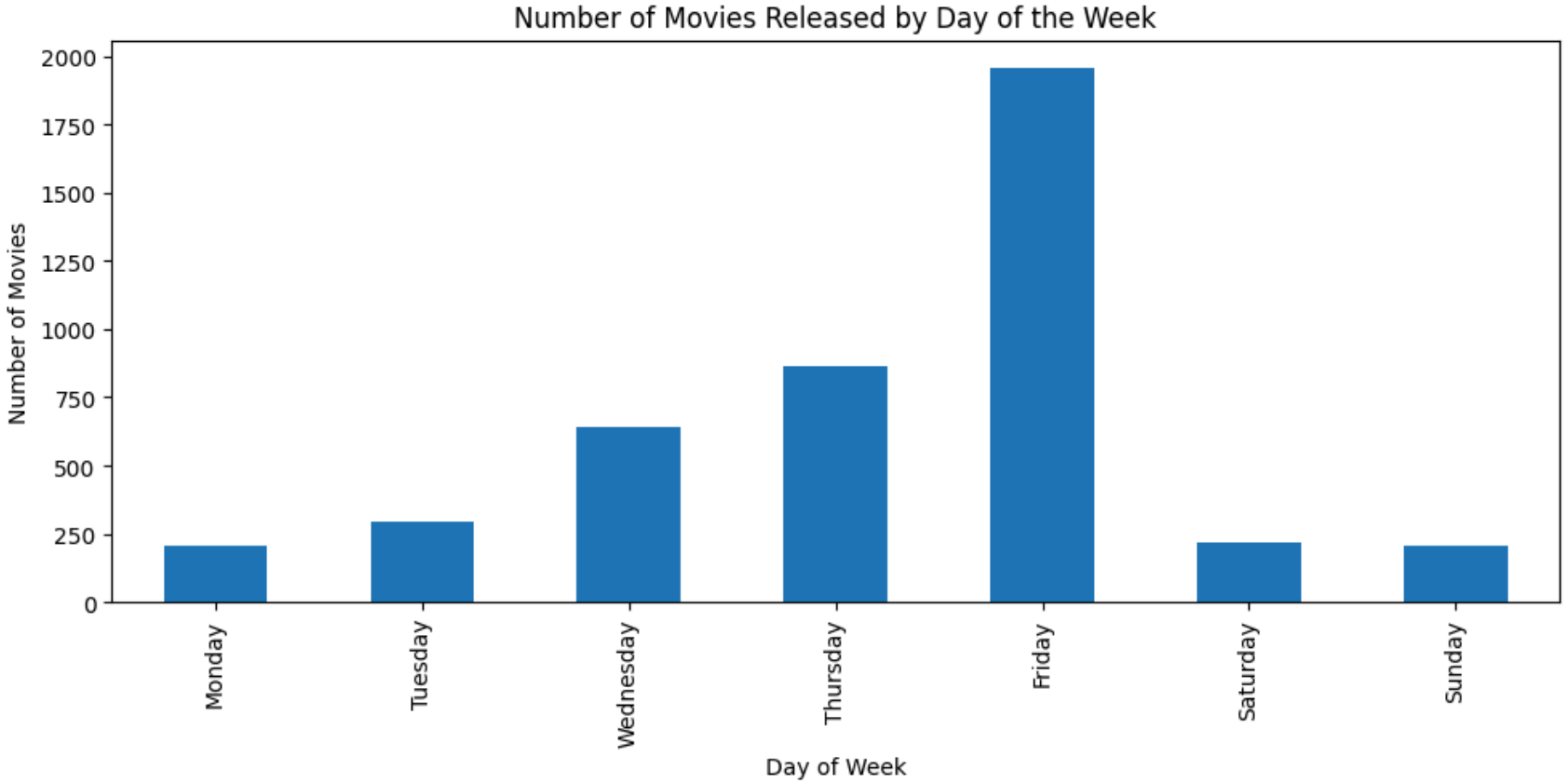
In [ ]:
```
# convert the datetime column to datetime using pandas
clean_df['release_date'] = pd.to_datetime(clean_df['release_date'], errors='coerce')


# create year, month and day of week columns using .dt
clean_df["year"] = clean_df["release_date"].dt.year
clean_df["month"] = clean_df["release_date"].dt.month
clean_df["day_of_week"] = clean_df["release_date"].dt.day_name()

clean_df.to_csv("clean_data.csv", index=False)

movies_per_year = clean_df.groupby('year').size()

movies_per_month = clean_df.groupby('month').size()

movies_per_day = clean_df.groupby('day_of_week').size()

# Make sure day_of_week is ordered
ordered_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
movies_per_day = movies_per_day.reindex(ordered_days)

fig, axs = plt.subplots(3, 1, figsize=(10, 15))

# Plot Movies by Year
movies_per_year.plot(ax=axs[0], kind='line', marker='o')
axs[0].set_title('Number of Movies Released By Year')
axs[0].set_xlabel('Year')
axs[0].set_ylabel('Number of Movies')

movies_per_month.index = movies_per_month.index.map(lambda x: calendar.month_name[int(x)] if pd.notnull(x) else 'Unkr


# Plot Movies by Month
movies_per_month.plot(ax=axs[1], kind='bar')
axs[1].set_title('Number of Movies Released By Month')
axs[1].set_xlabel('Month')
axs[1].set_ylabel('Number of Movies')
```

```
# Plot Movies by Day of the Week
movies_per_day.plot(ax=axs[2], kind='bar')
axs[2].set_title('Number of Movies Released by Day of the Week')
axs[2].set_xlabel('Day of Week')
axs[2].set_ylabel('Number of Movies')

plt.tight_layout()

plt.show()
```


Number of Movies Released By Year
Number of Movies
250
200
150
100
50
0
1920
1940
1960
1980
2000
2020
Year


Number of Movies Released By Month
Number of Movies
500
400
300
200
100





Analysis

From the Graphs above we can see that the number of movies which released started rapdily increasing over the years, especially around the late 1900s to early 2000s peaking at around 250 movies per year which then came to a crash 2019-2020 which was around the rise of COVID.From the months, we can that more movies tend to be released during the fall which was unexpected, as I thought the peak time for movies would be around the summer or maybe the holidays. As for days of the week, Fridays is the most

common release date for movies coming in at around 2000 which is to be expected as its the end of the week and most likely the time when everyone is free.

Importing libraries and a function for fixing spoken languages

In [ ]:
```
def fix_lang(column_data):
  for i in range(len(iso_639_1_languages)):
    if column_data == list(iso_639_1_languages.keys())[i]:
      return list(iso_639_1_languages.values())[i]
  return ""
```

Extract functions

In [ ]:
```
def extract_name(column_data):
  try:
    data = ast.literal_eval(column_data)
    names = []
    for d in data:
      if 'name' in d:
        names.append(d['name'])
    return ", ".join(names)
  except:
    return ""

def extract_id(column_data):
  try:
    data = ast.literal_eval(column_data)

    ids = []
    for d in data:
      if 'id' in d:
        ids.append(str(d['id']))
    return ", ".join(ids)
  except:
    return ""
```

```
def extract_iso_3166(column_data):
  try:
    data = ast.literal_eval(column_data)
    iso3166 = []
    for d in data:
      if 'iso_3166_1' in d:
        iso3166.append(d['iso_3166_1'])
      return ", ".join(iso3166)
  except:
      return ""

def extract_iso_639(column_data):
  try:
    data = ast.literal_eval(column_data)
    iso639 = []
    for d in data:
      if 'iso_639_1' in d:
        iso639.append(d['iso_639_1'])
    return ", ".join(iso639)
  except:
    return ""
```

Extracting info from cleaned data

From the following columns : genres, keywords, production_companies, production_countries, and spoken_languages

In [ ]:
```
#genres
genre_name, genre_id = [], []

for i in clean_df['genres']:
  genre_name.append(extract_name(i))
  genre_id.append(extract_id(i))


#keywords
keyword_name, keyword_id = [], []

for i in clean_df['keywords']:
```

```
  keyword_name.append(extract_name(i))
  keyword_id.append(extract_id(i))


#production_company
company_name, company_id = [], []
for i in clean_df['production_companies']:
  company_name.append(extract_name(i))
  company_id.append(extract_id(i))


#production_country
country_name, country_iso = [], []
for i in clean_df['production_countries']:
  country_name.append(extract_name(i))
  country_iso.append(extract_iso_3166(i))


#spoken_languages
lang_name, lang_iso = [], []
for i in clean_df['spoken_languages']:
  names = []
  iso = []
  try:
    data = ast.literal_eval(i)
    for d in data:
      if 'name' in d and 'iso_639_1' in d:
        name = fix_lang(d['iso_639_1'])
        names.append(name)
        iso.append(d['iso_639_1'])
  except:
      pass
  lang_name.append(", ".join(names))
  lang_iso.append(", ".join(iso))
```

csv files for each of those colums with the movie title

In [ ]:
```
#generating dataframes for each of those columns
genre_df = clean_df[['title']].copy()
genre_df['genre_name'] = genre_name
genre_df['genre_id'] = genre_id
```

```
keyword_df = clean_df[['title']].copy()
keyword_df['keyword_name'] = keyword_name
keyword_df['keyword_id'] = keyword_id

production_company_df = clean_df[['title']].copy()
production_company_df['production_company_name'] = company_name
production_company_df['production_company_id'] = company_id

production_country_df = clean_df[['title']].copy()
production_country_df['production_country_name'] = country_name
production_country_df['production_country_iso'] = country_iso

spoken_language_df = clean_df[['title']].copy()
spoken_language_df['spoken_language_name'] = lang_name
spoken_language_df['spoken_language_iso'] = lang_iso

#writing csvs for the features
genre_df.to_csv("genre.csv", index=False)
keyword_df.to_csv("keyword.csv", index=False)
production_company_df.to_csv("production_company.csv", index=False)
production_country_df.to_csv("production_country.csv", index=False)
spoken_language_df.to_csv("spoken_language.csv", index=False)
```

Reformatting actors, characters and crew, roles

In [ ]:
```
credits_df = pd.read_csv('tmdb_5000_credits.csv')

credits_df['cast'] = credits_df['cast'].apply(ast.literal_eval)
credits_df['crew'] = credits_df['crew'].apply(ast.literal_eval)

cast_rows = []
for _, row in credits_df.iterrows():
  movie_id = row['movie_id']
  title = row['title']
  for actor in row['cast']:
    cast_rows.append({'movie_id': movie_id, 'title': title, 'actor_name': actor.get('name'), 'character': actor.get(

crew_rows = []
for _, row in credits_df.iterrows():
```

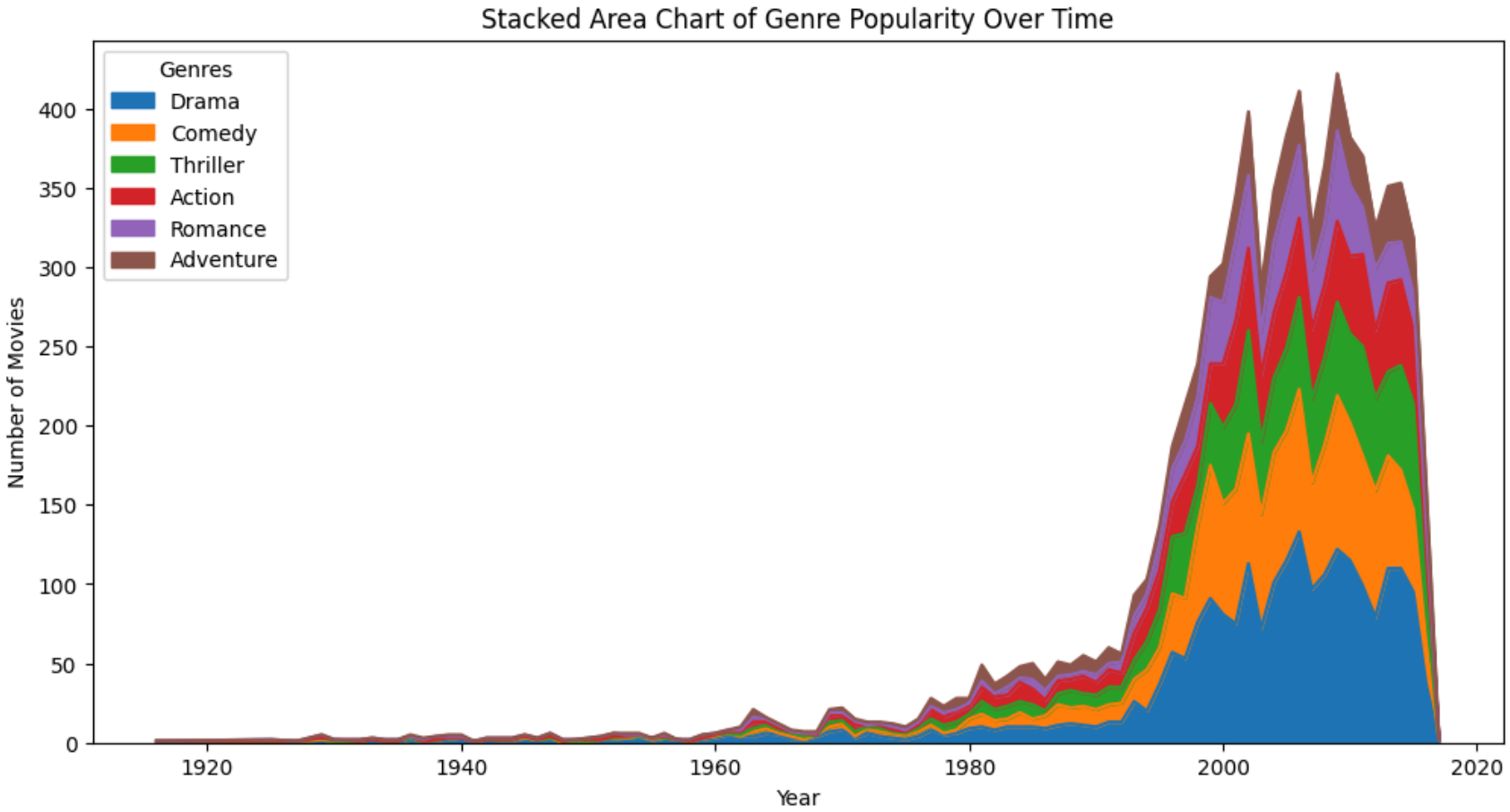
```
  movie_id = row['movie_id']
  title = row['title']
  for crew_member in row['crew']:
    crew_rows.append({'movie_id': movie_id,'title': title,'crew_name': crew_member.get('name'),'department': crew_mer

cast_df = pd.DataFrame(cast_rows)
crew_df = pd.DataFrame(crew_rows)

cast_df.to_csv('cast.csv', index=False)
crew_df.to_csv('crew.csv', index=False)
```

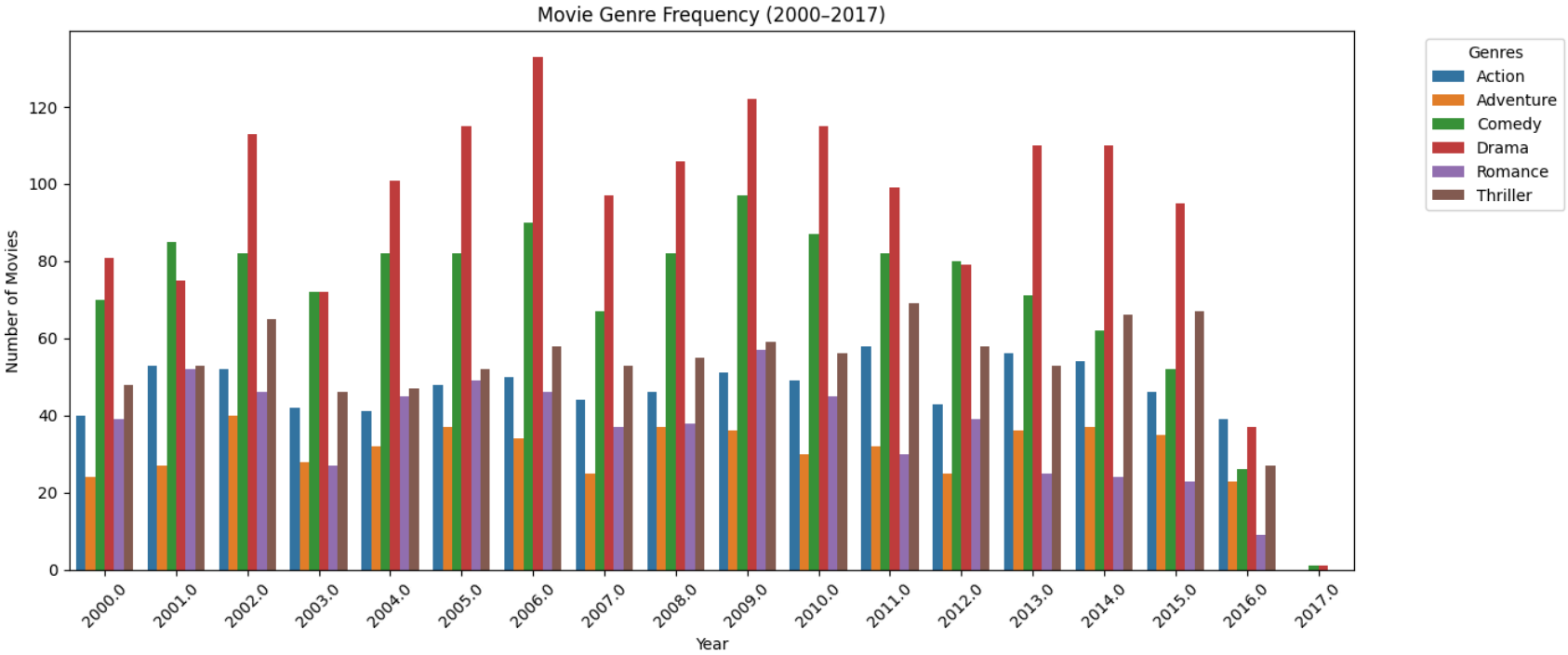
Discovering Movie Genre Trends

Trying to answer : Which genres are becoming more popular over the years?

In [ ]:
```
#load datasets
df = pd.read_csv("part1_movies.csv")
#Parsing release date to extract year
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
df['year'] = df['release_date'].dt.year

#Convert genres from string to list of dicts
df['genres'] = df['genres'].apply(lambda x: ast.literal_eval(x) if pd.notnull(x) else [])

#Extract only genre names
df['genre_names'] = df['genres'].apply(lambda genre_list: [g['name'] for g in genre_list])

#Explode to one genre per row
exploded = df.explode('genre_names')

#Group by year and genre and count occurrences
genre_trends = exploded.groupby(['year', 'genre_names']).size().reset_index(name='count')
#Stacked Area Chart plot
top_genres = genre_trends.groupby('genre_names')['count'].sum().sort_values(ascending=False).head(6).index
filtered = genre_trends[genre_trends['genre_names'].isin(top_genres)]
pivot = genre_trends.pivot(index='year', columns='genre_names', values='count').fillna(0)
pivot[top_genres].plot.area(figsize=(12, 6), stacked=True)
plt.title("Stacked Area Chart of Genre Popularity Over Time")
plt.xlabel("Year")
```

```
plt.ylabel("Number of Movies")
plt.legend(title="Genres")
plt.show()
print("\n")

#Filter b/w years 2000 to 2017
genre_2000_2017 = genre_trends[(genre_trends['year'] >= 2000) & (genre_trends['year'] <= 2017)]

#Top genres
top_genres = genre_2000_2017.groupby('genre_names')['count'].sum().sort_values(ascending=False).head(6).index
filtered = genre_2000_2017[genre_2000_2017['genre_names'].isin(top_genres)]

#Plot grouped bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=filtered, x='year', y='count', hue='genre_names')
plt.title('Movie Genre Frequency (2000-2017)')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
plt.legend(title='Genres', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


Stacked Area Chart of Genre Popularity Over Time
Genres
Drama
Comedy
Thriller
Action
Romance
Adventure
Number of Movies
0
50
100
150
200
250
300
350
400
1920
1940
1960
1980
2000
2020
Year



Which genres are becoming more popular over the years?

The visualizations reveal certain patterns about how popularity of film genres have evolved through time. The stacked area chart, spanning over a century, shows a sudden increase in overall film production beginning in the late 1980s, with peak volume between 2000 and 2015. Among the most popular genres are Drama, Comedy, and Action, with Drama experiencing the biggest percentage of film releases over the period. Notably, Thriller, Romance, and Adventure also demonstrate a steady rise in depiction, contributing to a huge percentage of the total number of films in more recent decades.

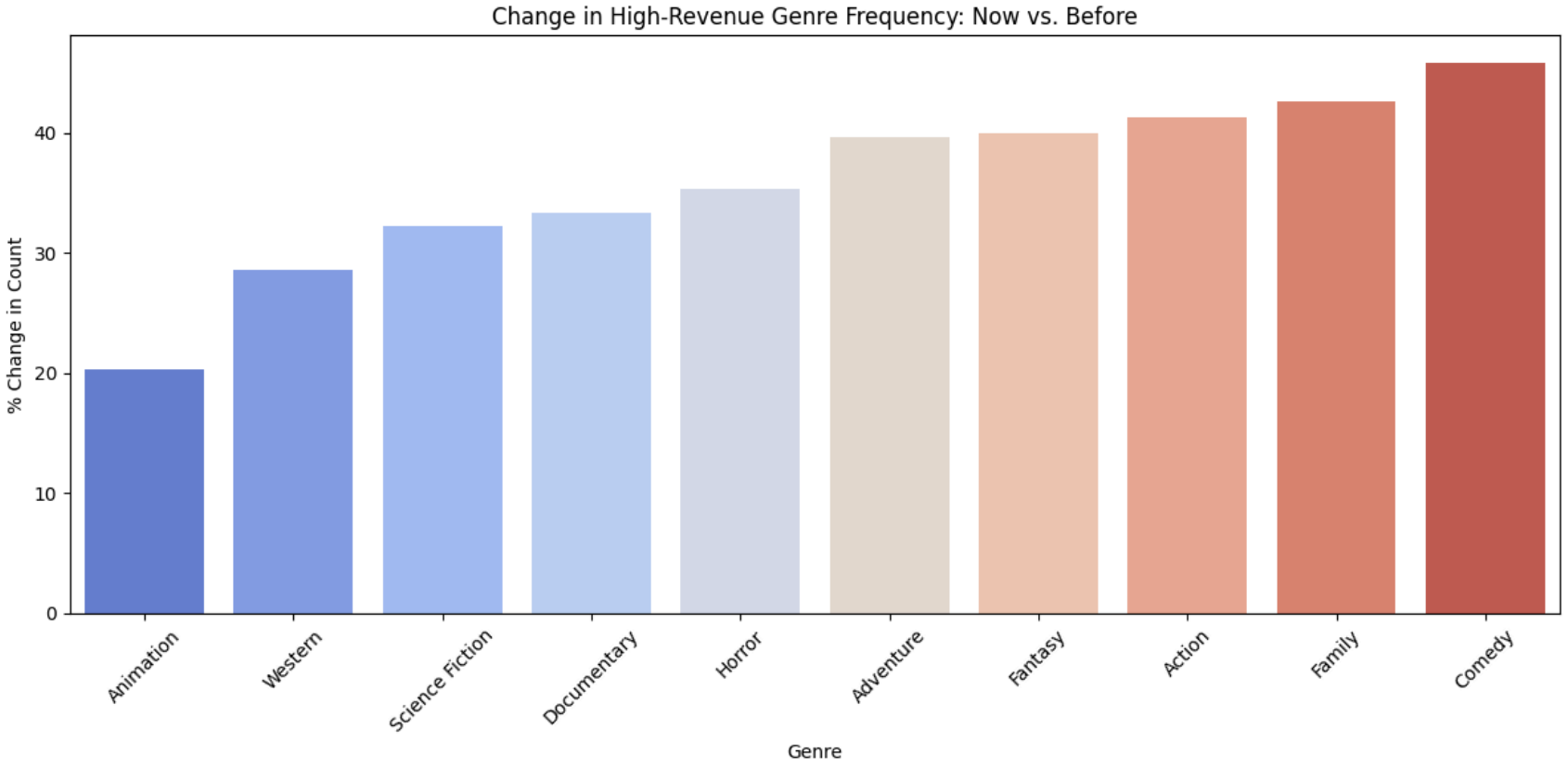
The grouped bar chart for the period 2000 to 2017 displays an indepth view of genre frequency in the modern era. For this period, Drama has the highest annual movie quantity, followed by Comedy and Thriller. Action and Adventure maintain steady but relatively lower production figures, whereas Romance maintains a consistent but mediocre presence. In general, the trend is that while Drama has remained consistent, other categories like Action and Thriller have picked up more in the recent past, possibly due to the fact that these have been featured in mass-market international blockbusters. These plots overall show how audience preferences and industry production have become diversified through time, especially in the 21st century.

DISCOVERING MOVIE GENRE TRENDS PART2.

Trying to answer: Do certain genres show more often in high-revenue movies now than before?

In [ ]:
```
#Splitting the data at 2010
now = df[df['year'] >= 2010].copy()
before = df[df['year'] < 2010].copy()

high_revenue_cutoff = df['revenue'].quantile(0.75)
now_high = now[now['revenue'] >= high_revenue_cutoff]
before_high = before[before['revenue'] >= high_revenue_cutoff]
# Explode genre lists into rows
now_high_exploded = now_high.explode('genre_names')
before_high_exploded = before_high.explode('genre_names')


now_counts = now_high_exploded['genre_names'].value_counts().reset_index()
now_counts.columns = ['genre', 'count_now']

before_counts = before_high_exploded['genre_names'].value_counts().reset_index()
before_counts.columns = ['genre', 'count_before']


merged = pd.merge(before_counts, now_counts, on='genre', how='outer').fillna(0)

# Optional: Add percentage change
merged['change_%'] = ((merged['count_now'] - merged['count_before']) / merged['count_before'].replace(0, 1)) * 100

# Sort by change
merged_sorted = merged.sort_values(by='change_%', ascending=False)

top_change = merged_sorted.head(10).copy()
top_change['change_%'] *= -1

plt.figure(figsize=(12, 6))
sns.barplot(data=top_change, x='genre', y='change_%', hue='genre', palette='coolwarm', legend=False)
plt.title('Change in High-Revenue Genre Frequency: Now vs. Before')
plt.ylabel('% Change in Count')
plt.xlabel('Genre')
```

```
plt.xticks(rotation=45)
plt.axhline(0, color='black', linewidth=0.8)
plt.tight_layout()
plt.show()
```



Do certain genres show more often in high-revenue movies now than before?

The chart shows the variation in how often various genres are found in high-grossing films in the time since 2010 versus prior to that. Particularly, the genres of Comedy, Family, and Action, which have experienced the largest growth, indicating a definitive trend toward more universally appealing, high-entertainment fare in the film industry. These genres are often associated with big franchises and global box office blockbusters, so it may not be a coincidence that they are rapidly taking over the blockbuster

terrain. Other genres such as Fantasy, Adventure, and even Horror also see more presence, reflecting greater viewer demand for visionary and emotionally powerful storytelling. Interestingly, even Documentary and Science Fiction genres have been more recognized under the high-revenue category, perhaps due to enhanced streaming activities and diversified TV watching.

To measure this relationship, the percentage change in count was approximated by using the formula:

$$\text{Change } \% = \left(\frac{\text{count}_{\text{now}} - \text{count}_{\text{before}}}{\text{count}_{\text{before}}} \right) \times 100$$

This method provides a comparison of the count of high-revenue films for all the genres before and after 2010 with the earlier phase. It was used because it normalizes the difference by the original count, allowing us to see which genres grew proportionally the most, regardless of their starting size. This approach highlights not just which genres are more common now, but which ones have experienced the largest growth in representation, providing clearer insight into evolving industry trends and audience preferences.

Data Cleaning Continuation

Part 2: In this section, we're mainly cleaning up the columns which had originally contained dictionaries or lists of dictionaries such as "genres", "production company", "production country", and "spoken langauge". We first load in the csv files in as dataframes, get all the titles of the movies which have Nan values in those respective fields such as genre, production company and so on - then we send each of those titles into the OMDb API and get the missing data from the response. If the data was in the API response, we then update the dataframe for the movie accordingly as a way to get rid of all the Nan values. Any data that was not found through the API was found manually and inserted into the dataframe

In [ ]:
```
genre = pd.read_csv("genre.csv")
nan_genre = genre[genre['genre_name'].isna()]['title']
nan_titles = nan_genre.to_list()



for title in nan_titles:
  dt = get_movie_details(title, "16f3e56e")

  if dt:
```

```
    genre.loc[genre['title'] == title, 'genre_name'] = dt["Genre"]

  else:

    print(f"Unable to find information for movie: {title}")
```

```
Unable to find information for movie: America Is Still the Place
Unable to find information for movie: The Blood of My Brother: A Story of Death in Iraq
```

In [ ]:
```
"""
Fix up the remaining genres
"""

genre.loc[genre['title'] == 'America Is Still the Place', 'genre_name'] = 'Drama'
genre.loc[genre['title'] == 'The Blood of My Brother: A Story of Death in Iraq', 'genre_name'] = 'Documentary, War'

genre.to_csv('fixed_genres.csv', index=False)
```

In [ ]:
```
production_country_iso = {
    "USA": "US",
    "United States": "US",
    "United States of America": "US",
    "UK": "GB",
    "United Kingdom": "GB",
    "Canada": "CA",
    "France": "FR",
    "Germany": "DE",
    "Italy": "IT",
    "Spain": "ES",
    "Australia": "AU",
    "New Zealand": "NZ",
    "Japan": "JP",
    "China": "CN",
    "Hong Kong": "HK",
    "South Korea": "KR",
    "India": "IN",
    "Mexico": "MX",
    "Brazil": "BR",
    "Argentina": "AR",
    "Sweden": "SE",
    "Denmark": "DK",
    "Norway": "NO",
```

```
    "Finland": "FI",
    "Belgium": "BE",
    "Netherlands": "NL",
    "Ireland": "IE",
    "Russia": "RU",
    "Poland": "PL",
    "Portugal": "PT",
    "Switzerland": "CH",
    "Czech Republic": "CZ",
    "Austria": "AT",
    "Hungary": "HU",
    "Greece": "GR",
    "Turkey": "TR",
    "Thailand": "TH",
    "Philippines": "PH",
    "Indonesia": "ID",
    "Malaysia": "MY",
    "Singapore": "SG",
    "Taiwan": "TW",
    "South Africa": "ZA",
    "Egypt": "EG",
    "Israel": "IL",
    "Iran": "IR",
    "Romania": "RO",
    "Bulgaria": "BG",
    "Slovakia": "SK",
    "Iceland": "IS",
    "Luxembourg": "LU",
    "Latvia": "LV",
    "Lithuania": "LT",
    "Estonia": "EE",
    "Ukraine": "UA",
    "Croatia": "HR",
    "Serbia": "RS",
    "Slovenia": "SI",
    "Chile": "CL",
    "Colombia": "CO",
    "Venezuela": "VE",
    "Peru": "PE",
    "Uruguay": "UY",
    "Cuba": "CU",
    "Morocco": "MA",
```

```
    "Algeria": "DZ",
    "Tunisia": "TN",
    "Pakistan": "PK",
    "Bangladesh": "BD",
    "Vietnam": "VN",
    "Nepal": "NP",
    "Sri Lanka": "LK"
}
```

In [ ]:
```
p_country = pd.read_csv("production_country.csv")

countries_title = p_country[p_country['production_country_name'].isna()]['title']

for title in countries_title:
  dt = get_movie_details(title, "16f3e56e")

  if dt:

    p_country.loc[p_country['title'] == title, 'production_country_name'] = dt["Country"]

    first_country = dt["Country"].split(",")[0].strip()

    country_code = production_country_iso[first_country]

    p_country.loc[p_country['title'] == title, 'production_country_iso'] = country_code


  else:

    print(f"Unable to find Country information for movie: {title}")
```

```
Unable to find Country information for movie: Dreamer: Inspired By a True Story
Unable to find Country information for movie: Betty Fisher and Other Stories
Unable to find Country information for movie: Saving Private Perez
Unable to find Country information for movie: Last I Heard
Unable to find Country information for movie: Down & Out With The Dolls
Unable to find Country information for movie: The Blade of Don Juan
Unable to find Country information for movie: America Is Still the Place
Unable to find Country information for movie: The Blood of My Brother: A Story of Death in Iraq
```

In [ ]:
```
"""
Fix up the remaining production countries
```

```
"""

p_country.loc[p_country['title'] == 'Dreamer: Inspired By a True Story', ['production_country_name', 'production_cou
p_country.loc[p_country['title'] == 'Betty Fisher and Other Stories', ['production_country_name', 'production_country
p_country.loc[p_country['title'] == 'Saving Private Perez', ['production_country_name', 'production_country_iso']] =
p_country.loc[p_country['title'] == 'Last I Heard', ['production_country_name', 'production_country_iso']] = ['Unitec
p_country.loc[p_country['title'] == 'Down & Out With The Dolls', ['production_country_name', 'production_country_iso
p_country.loc[p_country['title'] == 'The Blade of Don Juan', ['production_country_name', 'production_country_iso']] =
p_country.loc[p_country['title'] == 'America Is Still the Place', ['production_country_name', 'production_country_isc
p_country.loc[p_country['title'] == 'The Blood of My Brother: A Story of Death in Iraq', ['production_country_name',


p_country.to_csv('fixed_countries.csv', index=False)
```

In [ ]:
```
s_language = pd.read_csv("spoken_language.csv")

nan_languages = s_language[s_language['spoken_language_name'].isna()]['title']

for title in nan_languages:
  dt = get_movie_details(title, "16f3e56e")

  if dt:

    s_language.loc[s_language['title'] == title, 'spoken_language_name'] = dt["Language"]

  else:

    print(f"Unable to find Language information for movie: {title}")
```

```
Unable to find Language information for movie: VeggieTales: The Pirates Who Don't Do Anything
Unable to find Language information for movie: Running Forever
Unable to find Language information for movie: Saving Private Perez
Unable to find Language information for movie: To Be Frank, Sinatra at 100
Unable to find Language information for movie: Down & Out With The Dolls
Unable to find Language information for movie: America Is Still the Place
Unable to find Language information for movie: The Blood of My Brother: A Story of Death in Iraq
```

In [ ]:
```
"""
Fix up the remaining spoken languages
"""

s_language.loc[s_language['title'] == "VeggieTales: The Pirates Who Don't Do Anything", 'spoken_language_name'] = 'En
```

```
s_language.loc[s_language['title'] == 'Running Forever', 'spoken_language_name'] = 'English'
s_language.loc[s_language['title'] == 'Saving Private Perez', 'spoken_language_name'] = 'Spanish'
s_language.loc[s_language['title'] == 'To Be Frank, Sinatra at 100', 'spoken_language_name'] = 'English'
s_language.loc[s_language['title'] == 'Down & Out With The Dolls', 'spoken_language_name'] = 'English'
s_language.loc[s_language['title'] == 'America Is Still the Place', 'spoken_language_name'] = 'English'
s_language.loc[s_language['title'] == 'The Blood of My Brother: A Story of Death in Iraq', 'spoken_language_name'] =

# drop the iso language column
s_language.drop(columns='spoken_language_iso', inplace=True)

s_language['num_languages'] = s_language['spoken_language_name'].apply(lambda x: len(x.split(',')))

# Write to CSV
s_language.to_csv('fixed_languages.csv', index=False)
```

In [ ]:

```
p_company = pd.read_csv("production_company.csv")

missing_companies = p_company[p_company['production_company_name'].isna()]

m1 = pd.read_csv("m1.csv")
m2 = pd.read_csv("m2.csv")
m3 = pd.read_csv("m3.csv")

m2 = m2.rename(columns={'production_companies': 'production_company_name'})
m3 = m3.rename(columns={'production_companies': 'production_company_name'})

m1 = pd.concat([m1, m2, m3], ignore_index=True)

# Merge only on rows where p_company is missing
filled = p_company.merge(
    m1,
    on='title',
    how='left',
    suffixes=('', '_new')
)

# Fill NaNs in original column with values from m1
filled['production_company_name'] = filled['production_company_name'].fillna(filled['production_company_name_new'])

# Drop the helper column
filled = filled.drop(columns=['production_company_name_new', 'production_company_id'])
```

```
filled.to_csv("cleaned_companies.csv")

num_missing = filled['production_company_name'].isna().sum()
print(f"Number of missing production company names: {num_missing}")

missing_rows = filled[filled['production_company_name'].isna()]
print(missing_rows)
```

```
Number of missing production company names: 8
                          title production_company_name  \
4170               American Desi                      NaN
4173  Love and Other Catastrophes                      NaN
4177              Teeth and Blood                      NaN
4179                   The Motel                      NaN
4184   The Sticky Fingers of Time                      NaN
4186                         Rust                      NaN
4190                   UnDivided                      NaN
4262        Dude Where's My Dog?                      NaN

     production_company_id
4170                   NaN
4173                   NaN
4177                   NaN
4179                   NaN
4184                   NaN
4186                   NaN
4190                   NaN
4262                   NaN
```

Correlation Analysis

Merging all key factors into one analysis dataframe, and calculating frequencies

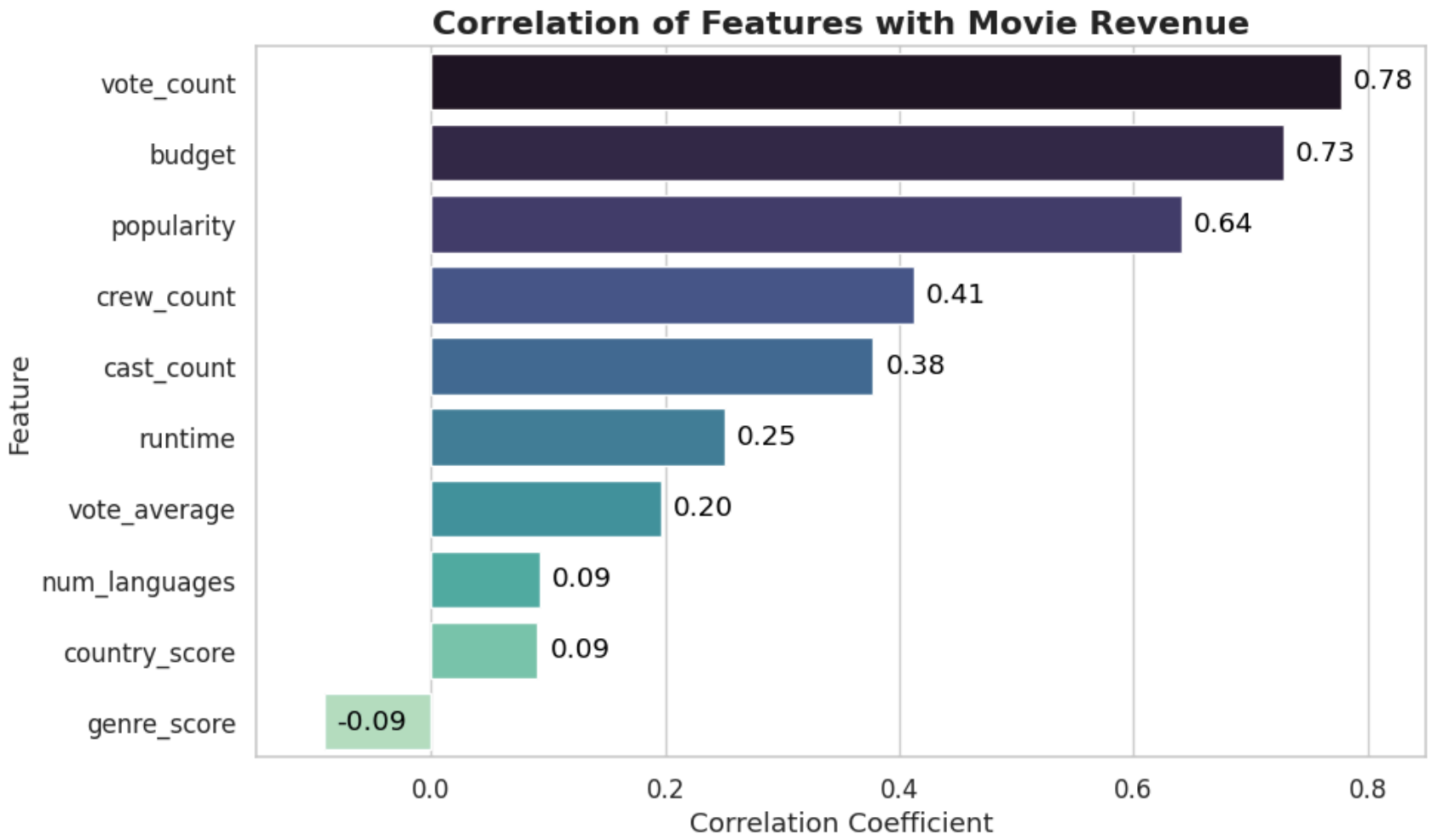
In [ ]:
```
movies_df = pd.read_csv('part1_movies.csv')
genres_df = pd.read_csv('fixed_genres.csv')
languages_df = pd.read_csv('fixed_languages.csv')
countries_df = pd.read_csv('fixed_countries.csv')
cast_df = pd.read_csv('cast.csv')
crew_df = pd.read_csv('crew.csv')
```

```
#merging genre,language and country
merged_df = movies_df.merge(genres_df[['title', 'genre_name']], on='title', how='left')
merged_df = merged_df.merge(languages_df[['title', 'num_languages']], on='title', how='left')
merged_df = merged_df.merge(countries_df[['title', 'production_country_name']], on='title', how='left')

#calculating frequency
genre_counts = genres_df['genre_name'].value_counts().to_dict()
country_counts = countries_df['production_country_name'].value_counts().to_dict()
merged_df['genre_score'] = merged_df['genre_name'].map(genre_counts)
merged_df['country_score'] = merged_df['production_country_name'].map(country_counts)

analysis_df = merged_df[['budget', 'popularity', 'runtime', 'vote_average', 'vote_count', 'revenue', 'num_languages']
analysis_df['genre_score'] = merged_df['genre_score']
analysis_df['country_score'] = merged_df['country_score']
analysis_df = analysis_df.dropna(subset=['revenue'])

#merging cast and crew
cast_counts = cast_df.groupby('movie_id').size().reset_index(name='cast_count')
crew_counts = crew_df.groupby('movie_id').size().reset_index(name='crew_count')

#filtering director from crew
directors = crew_df[crew_df['job'] == 'Director']
director_names = directors.groupby('movie_id')['crew_name'].first().reset_index(name='director')

#merge with movie IDs and revenue
movie_features = movies_df[['id', 'title', 'revenue']].copy()
movie_features = movie_features.merge(cast_counts, left_on='id', right_on='movie_id', how='left')
movie_features = movie_features.merge(crew_counts, left_on='id', right_on='movie_id', how='left')
movie_features = movie_features.merge(director_names, left_on='id', right_on='movie_id', how='left')
movie_features['cast_count'] = movie_features['cast_count'].fillna(0)
movie_features['crew_count'] = movie_features['crew_count'].fillna(0)
movie_features['director'] = movie_features['director'].fillna('Unknown')

analysis_df['cast_count'] = movie_features.set_index('id').loc[merged_df['id']]['cast_count'].values
analysis_df['crew_count'] = movie_features.set_index('id').loc[merged_df['id']]['crew_count'].values
```

Correlation analysis into bar plot

In [ ]:
```
#correlation bar plot
correlation_series = analysis_df.corr(numeric_only=True)['revenue'].sort_values(ascending=False)
```

```
sns.set(style="whitegrid", font_scale=1.1)

plt.figure(figsize=(10, 6))

corr_bplot = sns.barplot(x=correlation_series.drop('revenue').values, y=correlation_series.drop('revenue').index, hue

plt.title('Correlation of Features with Movie Revenue', fontsize=16, fontweight='bold')
plt.xlabel('Correlation Coefficient')
plt.ylabel('Feature')
plt.xlim(-0.15, 0.85)

for i, v in enumerate(correlation_series.drop('revenue')):
  corr_bplot.text(v + 0.01, i, f"{v:.2f}", color='black', va='center')

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



This bar plot shows how movie features are correlated with the movie revenue, based on Pearson correlation coefficients.

High positive correlation:

Among all the features, vote count, budget, and popularity have the highest correlation with revenue which indicates that these features are the most reliable indicators for a higher revenue. This explains that movies usually generate higher revenues upon receiving more audience engagement in form of votes or popularity(online engagement), and higher budget(production quality, larger-scale promotion, and broader distribution).

Moderate correlation:

crew_count and cast_count shows moderate correlation, suggesting that larger productions with more people involved on and off screen tend to do better, but project scale alone isn't a strong determinant unless combined with budget or audience engagement.

Weaker correlation:

runtime and vote_average shows weaker correlations. Interestingly, a movie's average rating (vote_average) has a much smaller influence on revenue compared to how many people rate the film (vote_count), showing that audience volume matters more than critical acclaim. The number of languages spoken in a movie or how frequently its genres or production countries appear in the dataset, had very low correlation values.

Negative correlation:

Genre_score had a slightly negative correlation, which may indicate that that more common genres do not necessarily lead to higher revenue.

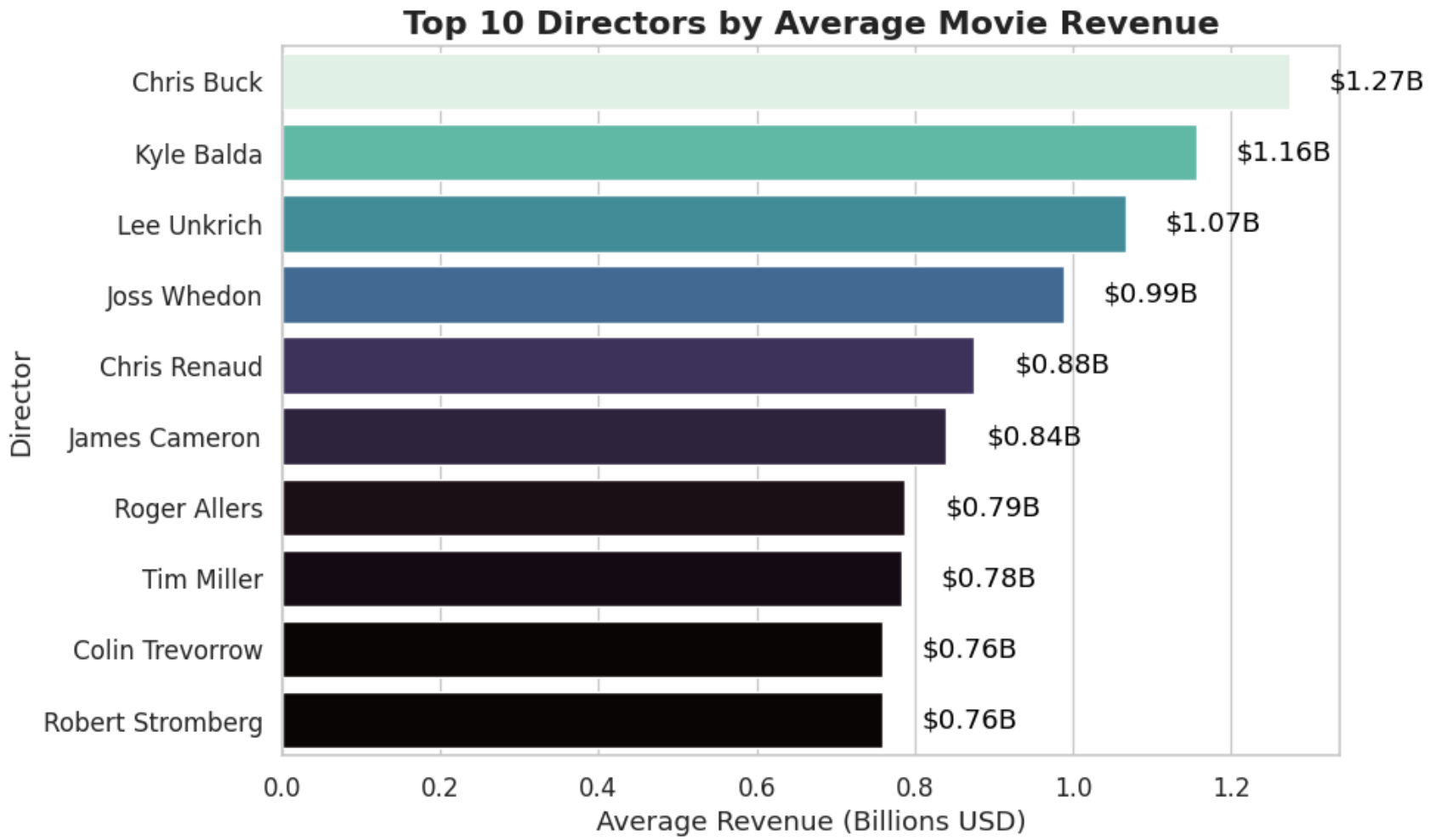
Overall, the analysis highlights that audience engagement, marketing, and higher budget on production resources are the most influential factors in determining a film's revenue performance.

Directors associated with higher revenues

This is not included in the correlation plot since this is a category(non-numeric):

In [ ]:
```
#top directors by revenue
top_directors = movie_features.groupby('director')['revenue'].mean().reset_index()
top_directors = top_directors.sort_values(by='revenue', ascending=False).head(10)

plt.figure(figsize=(10, 6))
director_plot = sns.barplot(x=top_directors['revenue']/1e9, y=top_directors['director'], hue=top_directors['revenue']
plt.title('Top 10 Directors by Average Movie Revenue', fontsize=16, fontweight='bold')
plt.xlabel('Average Revenue (Billions USD)')
plt.ylabel('Director')

for i, v in enumerate(top_directors['revenue'] / 1e9):
  director_plot.text(v + 0.05, i, f"${v:.2f}B", color='black', va='center')

plt.tight_layout()
```

```
plt.show()

# #top actors by revenue
# actor_revenue_df = cast_df.merge(movies_df[['id', 'revenue']], left_on='movie_id', right_on='id', how='left')
# top_actors_df = actor_revenue_df.groupby('actor_name')['revenue'].mean().reset_index()
# top_actors_df = top_actors_df.sort_values(by='revenue', ascending=False).head(10)

# plt.figure(figsize=(10, 6))
# plt.barh(top_actors_df['actor_name'], top_actors_df['revenue'] / 1e9)
# plt.title('Top 10 Actors by Average Movie Revenue')
# plt.xlabel('Average Revenue (Billions USD)')
# plt.gca().invert_yaxis()
# plt.grid(True)
# plt.tight_layout()
# plt.show()
```



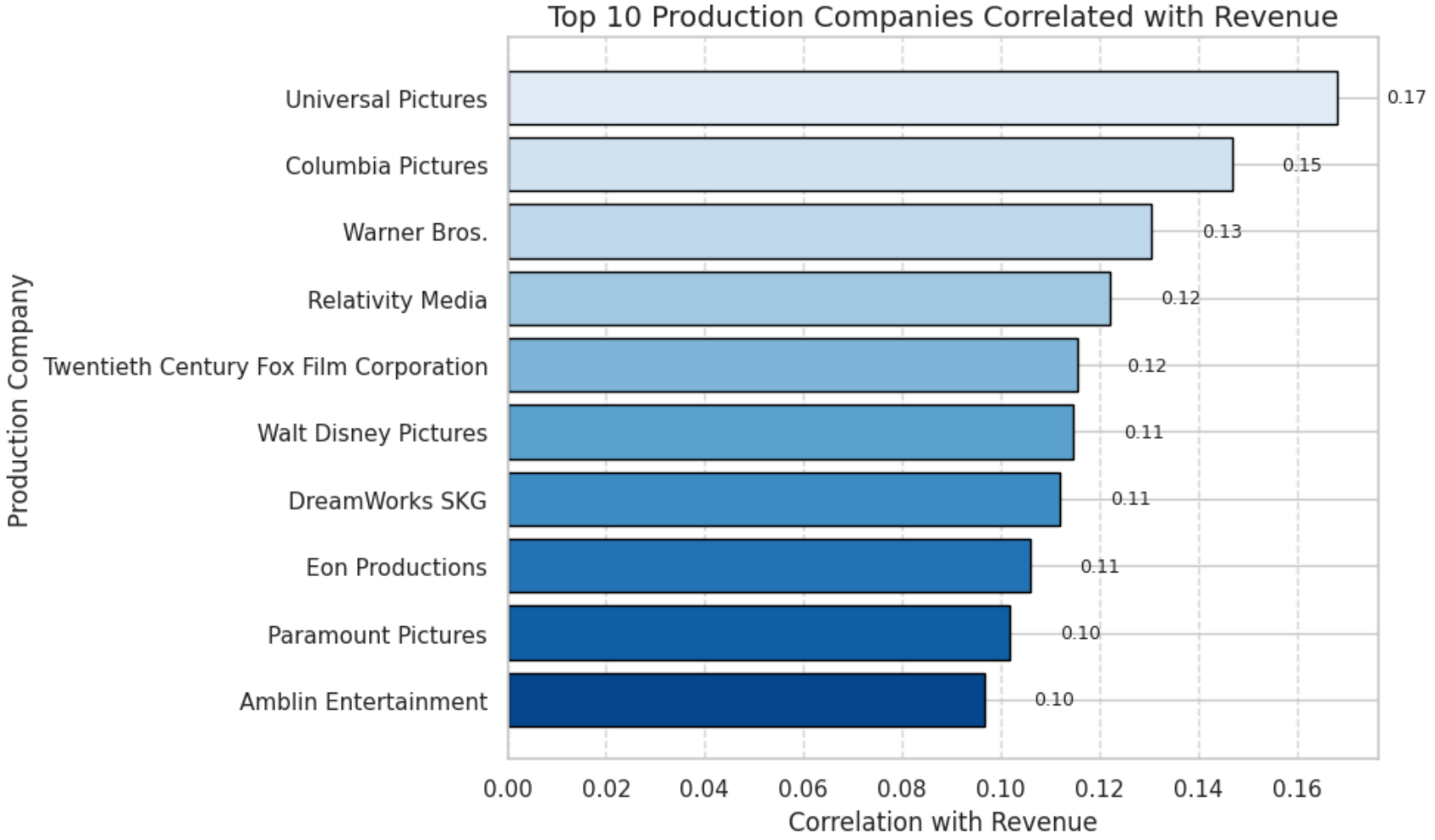
This shows the top ten directors ranked by the average revenue generated by the movies they directed.

Directors with a consistent track record of high-revenue films likely benefit from strong studio support, creative vision, and audience loyalty. Their presence can also attract major actors and high production investment, forming a feedback loop that increases a movie's chances of box office success.

Production companies associated with higher revenue

In [ ]:
```
#loading datasets
companies_df = pd.read_csv('part1_movies.csv')
revenue_df = pd.read_csv('cleaned_companies.csv')
merged_df = pd.merge(companies_df, revenue_df, on='title', how='inner')
#split the production companies into lists
#replace NaNs with an empty list
merged_df['company_list'] = merged_df['production_company_name'].fillna('').str.split(',')

#clean up whitespace
merged_df['company_list'] = merged_df['company_list'].apply(
  lambda x: [c.strip() for c in x if c.strip() != '']
)
#creates one-hot encoded company columns
mlb = MultiLabelBinarizer()
company_ohe = pd.DataFrame(mlb.fit_transform(merged_df['company_list']), columns=mlb.classes_,
                           index=merged_df.index)
#join one-hot columns to original data
final_df = pd.concat([merged_df, company_ohe], axis=1)
mlb = MultiLabelBinarizer()
company_ohe = pd.DataFrame(
  mlb.fit_transform(merged_df['company_list']),
  columns=mlb.classes_,
  index=merged_df.index
)

final_df = pd.concat([merged_df, company_ohe], axis=1)

#compute correlation
correlations = final_df[mlb.classes_].corrwith(final_df['revenue'])
correlations_sorted = correlations.sort_values(ascending=False)
sns.set(style="whitegrid")
top_companies = correlations_sorted.head(10)

#color gradient based on correlation values
colors = sns.color_palette("Blues", len(top_companies))
#Plot
plt.figure(figsize=(10, 6))
bars = plt.barh(top_companies.index, top_companies.values, color=colors, edgecolor='black')

#adding labels to each bar
for bar in bars:
```

```
  width = bar.get_width()
  plt.text(width + 0.01, bar.get_y() + bar.get_height()/2, f'{width:.2f}', va='center', fontsize=9)

# Formatting
plt.title('Top 10 Production Companies Correlated with Revenue', fontsize=14)
plt.xlabel('Correlation with Revenue', fontsize=12)
plt.ylabel('Production Company', fontsize=12)
plt.gca().invert_yaxis()  # Highest on top
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Creating Models

In [1]:
```
df = pd.read_csv('merged_with_directors.csv')

# define numeric features
num_feats = ['budget','popularity','runtime','vote_average','vote_count','num_languages']

mlb = MultiLabelBinarizer()

df['genres_list'] = df['genre_name'].str.split(r',\s*')

# One hot encodings for genres
genres_mh = pd.DataFrame(
  mlb.fit_transform(df['genres_list']),
  columns=[f"genre__{g}" for g in mlb.classes_],
  index=df.index
)

# One hot encodings for country
ohe_country = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
country_arr = ohe_country.fit_transform(df[['production_country_iso']])
country_cols = [f"country__{cat}" for cat in ohe_country.categories_[0]]
country_ohe = pd.DataFrame(country_arr, columns=country_cols, index=df.index)

# One hot encodings for Language
ohe_lang = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
lang_arr = ohe_lang.fit_transform(df[['spoken_language_name']])
lang_cols = [f"lang__{cat}" for cat in ohe_lang.categories_[0]]
lang_ohe = pd.DataFrame(lang_arr, columns=lang_cols, index=df.index)

# One hot encoding after finding 10 most popular
df['num_companies'] = df['production_company_name'].fillna('').str.count(',') + 1
top10 = (
    df['production_company_name']
      .str.split(r',\s*')
      .explode()
      .value_counts()
      .head(10)
      .index
```

```
)

def encode_top10(cell):
  comps = set(cell.split(r', ')) if pd.notna(cell) else set()
  return [int(c in comps) for c in top10]

prodco_ohe = pd.DataFrame(
  df['production_company_name'].apply(encode_top10).tolist(),
  columns=[f"prodCo__{c}" for c in top10],
  index=df.index
)

# create Data Matrix with numeric and categorical features
X = pd.concat([
    df[num_feats],
    genres_mh,
    country_ohe,
    lang_ohe,
    df[['num_companies']],
    prodco_ohe
], axis=1).fillna(0)

# drop duplicate columns
X = X.loc[:, ~X.columns.duplicated()]

# log transform budget due to power law
X['budget'] = np.log1p(df['budget'])

# log transform target revenue due to power law
y_log = np.log1p(df['revenue'].values)

# split data set to train and test set
X_train_raw, X_test_raw, y_train_log, y_test_log = train_test_split(
    X, y_log, test_size=0.2, random_state=42
)

# Map director mean to each split
global_mean = df['revenue'].mean()
director_means = df.loc[X_train_raw.index].groupby('director_name')['revenue'].mean()
X_train_raw['director_mean'] = df.loc[X_train_raw.index, 'director_name'].map(director_means)
X_test_raw['director_mean']  = df.loc[X_test_raw.index, 'director_name'].map(director_means)
X_train_raw['director_mean'] = X_train_raw['director_mean'].fillna(global_mean)
```

```
X_test_raw['director_mean']  = X_test_raw['director_mean'].fillna(global_mean)

# scaled features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_raw)
X_test_scaled  = scaler.transform(X_test_raw)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train_raw.columns, index=X_train_raw.index)
X_test_scaled  = pd.DataFrame(X_test_scaled, columns=X_test_raw.columns, index=X_test_raw.index)


# Define the models

models = {
  'GBM': GradientBoostingRegressor(
      n_estimators=300,
      learning_rate=0.05,
      max_depth=10,
      subsample=0.8,
      random_state=0
  ),
  'RandomForest': RandomForestRegressor(
      n_estimators=300,
      max_depth=15,
      min_samples_leaf=1,
      random_state=0
  ),
  'KNN (k=3)': KNeighborsRegressor(
      n_neighbors=3,
      weights='distance',
      metric='manhattan'
  )
}

results = []

# fit and evaluate for each model

for name, model in models.items():
  model.fit(X_train_scaled, y_train_log)
  y_pred_log = model.predict(X_test_scaled)
```

```
  # inverse log transform predictions
  y_pred = np.expm1(y_pred_log)
  y_true = df['revenue'].values[X_test_raw.index]

  rmse = np.sqrt(np.mean((y_true - y_pred)**2))
  r2   = r2_score(y_true, y_pred)
  results.append((name, rmse, r2))


print(f"{'Model':16} | {'RMSE':>10} | {'R^2':>6}")
print("-" * 50)
for name, rmse, r2 in results:
  print(f"{name:16} | {rmse:10,.0f} | {r2:6.3f}")
```

```
Model            |       RMSE |    R^2
--------------------------------------------------
GBM              | 43,626,769 |  0.369
RandomForest     | 43,916,663 |  0.360
KNN (k=3)        | 50,463,584 |  0.155
```

Models with K Fold Cross Validation

In [2]:
```
df = pd.read_csv('merged_with_directors.csv')

num_feats = ['budget','popularity','runtime','vote_average','vote_count','num_languages']

df['genres_list'] = df['genre_name'].str.split(r',\s*')
mlb = MultiLabelBinarizer()
genres_mh = pd.DataFrame(
  mlb.fit_transform(df['genres_list']),
  columns=[f"genre__{g}" for g in mlb.classes_],
  index=df.index
)

ohe_country = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
country_arr = ohe_country.fit_transform(df[['production_country_iso']])
country_cols = [f"country__{cat}" for cat in ohe_country.categories_[0]]
country_ohe = pd.DataFrame(country_arr, columns=country_cols, index=df.index)

ohe_lang = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
```

```
lang_arr = ohe_lang.fit_transform(df[['spoken_language_name']])
lang_cols = [f"lang__{cat}" for cat in ohe_lang.categories_[0]]
lang_ohe = pd.DataFrame(lang_arr, columns=lang_cols, index=df.index)

df['num_companies'] = df['production_company_name'].fillna('').str.count(',') + 1
top10 = (
    df['production_company_name']
      .str.split(r',\s*')
      .explode()
      .value_counts()
      .head(10)
      .index
)

def encode_top10(cell):
  comps = set(cell.split(r', ')) if pd.notna(cell) else set()
  return [int(c in comps) for c in top10]

prodco_ohe = pd.DataFrame(
  df['production_company_name'].apply(encode_top10).tolist(),
  columns=[f"prodCo__{c}" for c in top10],
  index=df.index
)

X = pd.concat([
    df[num_feats],
    genres_mh,
    country_ohe,
    lang_ohe,
    df[['num_companies']],
    prodco_ohe
], axis=1).fillna(0)

X = X.loc[:, ~X.columns.duplicated()]

y_log = np.log1p(df['revenue'].values)

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

models = {
  'GBM': GradientBoostingRegressor(
      n_estimators=300,
```

```
        learning_rate=0.05,
        max_depth=10,
        subsample=0.8,
        random_state=0
    ),
    'RandomForest': RandomForestRegressor(
        n_estimators=300,
        max_depth=15,
        min_samples_leaf=1,
        random_state=0
    ),
    'KNN (k=3)': KNeighborsRegressor(
        n_neighbors=3,
        weights='distance',
        metric='manhattan'
    )
}

results = {name: [] for name in models}

# K Fold Cross Validation
for fold, (train_idx, val_idx) in enumerate(kfold.split(X)):
    X_train = X.iloc[train_idx].copy()
    X_val   = X.iloc[val_idx].copy()
    y_train_log, y_val_log = y_log[train_idx], y_log[val_idx]

    global_mean = df['revenue'].mean()
    director_means = df.iloc[train_idx].groupby('director_name')['revenue'].mean()

    X_train['director_mean'] = df.iloc[train_idx]['director_name'].map(director_means)
    X_val['director_mean']   = df.iloc[val_idx]['director_name'].map(director_means)

    X_train['director_mean'] = X_train['director_mean'].fillna(global_mean)
    X_val['director_mean']   = X_val['director_mean'].fillna(global_mean)

    for name, model in models.items():
        model.fit(X_train, y_train_log)
        y_pred_log = model.predict(X_val)
        y_pred = np.expm1(y_pred_log)
        y_true = df['revenue'].values[val_idx]

        rmse = np.sqrt(np.mean((y_true - y_pred) ** 2))
```

```
    r2   = r2_score(y_true, y_pred)
    mape = mean_absolute_percentage_error(y_true, y_pred)

    results[name].append((rmse, r2, mape))
    print(f"[Fold {fold+1}] {name:12}: RMSE={rmse:,.0f}, R^2={r2:.3f}")

print("\nModel          Mean RMSE   | Mean R^2 ")
print("--------------------------------------------------")
for name, metrics in results.items():
  avg_rmse = np.mean([m[0] for m in metrics])
  avg_r2   = np.mean([m[1] for m in metrics])
  print(f"{name:12} | {avg_rmse:9,.0f} | {avg_r2:8.3f} ")
```

```
[Fold 1] GBM         : RMSE=42,871,370, R^2=0.390
[Fold 1] RandomForest: RMSE=44,056,203, R^2=0.356
[Fold 1] KNN (k=3)   : RMSE=57,347,594, R^2=-0.091
[Fold 2] GBM         : RMSE=42,561,816, R^2=0.399
[Fold 2] RandomForest: RMSE=43,101,708, R^2=0.383
[Fold 2] KNN (k=3)   : RMSE=52,929,828, R^2=0.070
[Fold 3] GBM         : RMSE=38,894,622, R^2=0.481
[Fold 3] RandomForest: RMSE=39,728,792, R^2=0.459
[Fold 3] KNN (k=3)   : RMSE=52,821,793, R^2=0.043
[Fold 4] GBM         : RMSE=40,827,106, R^2=0.480
[Fold 4] RandomForest: RMSE=41,615,814, R^2=0.460
[Fold 4] KNN (k=3)   : RMSE=53,517,624, R^2=0.106
[Fold 5] GBM         : RMSE=42,913,794, R^2=0.425
[Fold 5] RandomForest: RMSE=43,119,610, R^2=0.419
[Fold 5] KNN (k=3)   : RMSE=54,263,592, R^2=0.080

Model          Mean RMSE   | Mean R^2
--------------------------------------------------
GBM          | 41,613,742 |    0.435
RandomForest | 42,324,425 |    0.415
KNN (k=3)    | 54,176,086 |    0.042
```

In [ ]:

```
GBM              | 43,626,769 |  0.369
RandomForest     | 43,916,663 |  0.360
KNN (k=3)        | 50,463,584 |  0.155
```

Analysis

After doing K Fold Cross Validation we can see the RMSE for all the models has decreased except for KNN which has increased by about 4 million. We can also observe that the R^2 values for all of the models has increased, except for KNN again.