What is our research question? Potentially: What makes a movie go big at the Box Office/ what makes a movie successful?

Define function for getting data from API

```
import requests
import pandas as pd
import time

#OMDb API key
api_key = "6e2fc5e0"

# Base URL for OMDb API
base_url = "http://www.omdbapi.com/"

def fetch_movie_data(title, api_key):
    params = {
        "apikey": api_key,
        "t": title
    }
    response = requests.get(base_url, params=params)
    if response.status_code == 200:
        data = response.json()
        if data.get("Response") == "True":
            return data
        else:
            print(f"Error: {data.get('Error')}")
    else:
        print(f"HTTP Error: {response.status_code}")
    return None
```

Define function for getting data from API and make DataFrame from data

```
# Load the titles
titles_df = pd.read_csv("titles.csv")
titles = titles_df["title"].head(5000)

# Fetch movie data
movie_data = []
for title in titles:
    movie_data.append(fetch_movie_data(title, api_key))
    time.sleep(0.1)  # Rate limiting
```

```
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
```

```
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
Error: Movie not found!
```

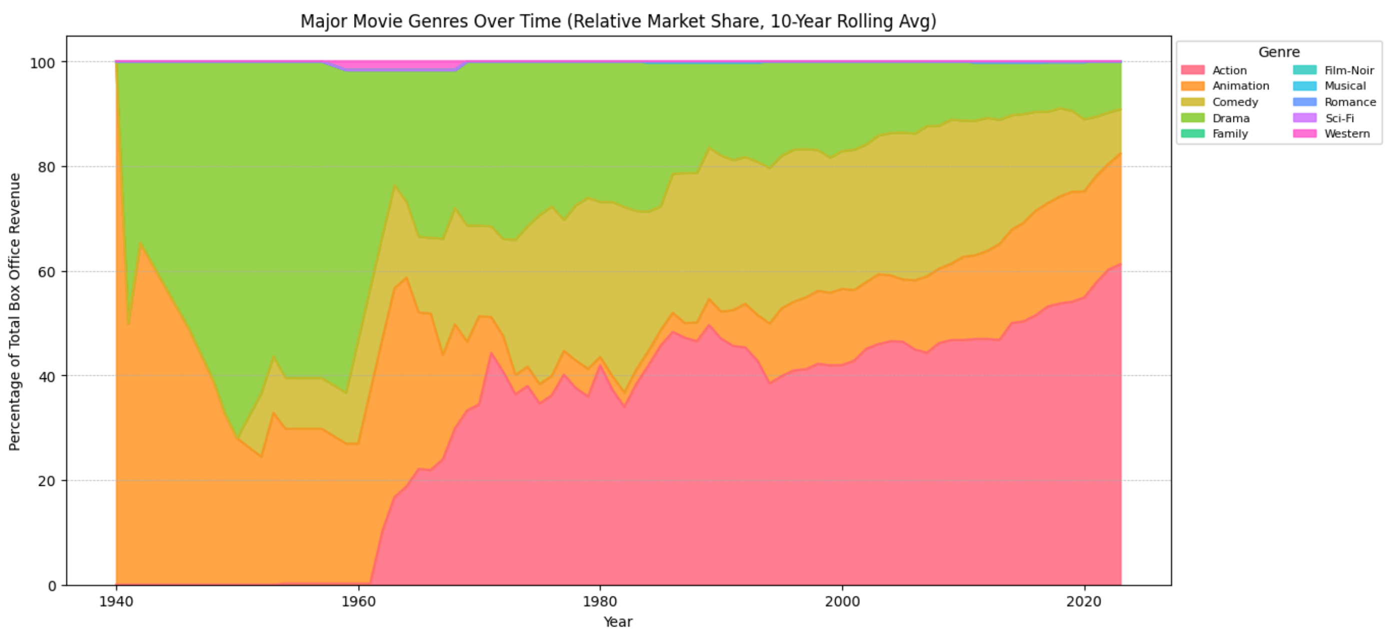
```
movie_data_filtered = [data for data in movie_data if data is not None]
movie_df_original = pd.DataFrame(movie_data_filtered)
movie_df_new = movie_df_original.drop(columns=["Writer", "Released", "Poster", "Type", "totalSeasons", "Response", "Website", "F
movie_df_new
```

Sam's Visualizations

Genre revenue over time as a percentage of the total

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

df = movie_df_new.copy()
df["Year"] = pd.to_numeric(df["Year"], errors="coerce")

df["BoxOffice"] = df["BoxOffice"].replace({'\$': '', ',': ''}, regex=True)
df["BoxOffice"] = pd.to_numeric(df["BoxOffice"], errors="coerce")

# Drop missing values
df = df.dropna(subset=["Genre", "BoxOffice", "Year"])

# Filter for reasonable years (avoid weird pre-1940 data)
df = df[df["Year"] >= 1940]

# Split genres into separate rows
df_exploded = df.assign(Genre=df["Genre"].str.split(", ")).explode("Genre")

# Standardize genre formatting (remove spaces, unify capitalization)
df_exploded["Genre"] = df_exploded["Genre"].str.strip().str.title()

# Aggregate revenue per genre
total_revenue_per_genre = df_exploded.groupby("Genre")["BoxOffice"].sum()

# Remove unwanted genres while **FORCING inclusion** of key classics
excluded_genres = {"Sport", "Crime", "Biography", "History", "Fantasy",
                   "Mystery", "Thriller", "Music", "Documentary", "Horror", "Tv Movie", "Adventure"}
included_genres = {"Western", "Film-Noir", "Musical"}  # Ensure these are included

# Ensure the classic genres are included even if not in the top revenue genres
top_genres = total_revenue_per_genre.nlargest(25).index
keep_genres = set(top_genres) - excluded_genres  # Remove unwanted genres
keep_genres.update(included_genres)  # **Force inclusion of Western, Film-Noir, Musical**

# **Manually add missing rows for "Film-Noir", "Western", and "Musical"**
for genre in included_genres:
    if genre not in df_exploded["Genre"].unique():
        df_exploded = pd.concat([df_exploded, pd.DataFrame({"Year": [1940], "Genre": [genre], "BoxOffice": [1]})], ignore_index=

df_exploded = df_exploded[df_exploded["Genre"].isin(keep_genres)]
```

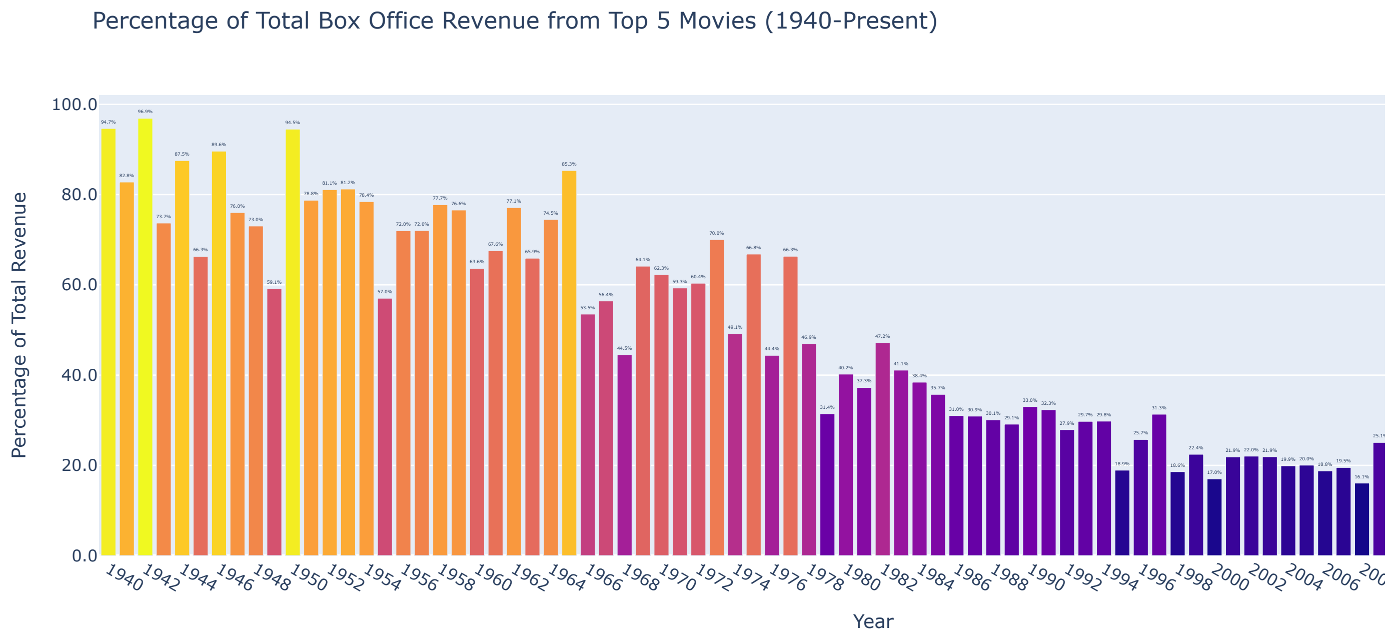
```
# Aggregate revenue per genre per year
genre_trends = df_exploded.groupby(["Year", "Genre"])["BoxOffice"].sum().reset_index()

# Pivot for better visualization
genre_pivot = genre_trends.pivot(index="Year", columns="Genre", values="BoxOffice").fillna(0)

# Compute each genre's percentage of total revenue per year
genre_pivot_relative = genre_pivot.div(genre_pivot.sum(axis=1), axis=0) * 100  # Convert to percentage

# Apply rolling average for smooth trends
roll = 10
genre_pivot_relative = genre_pivot_relative.rolling(window=roll, min_periods=1).mean()

# Generate a modified "husl" palette that is slightly **brighter but distinct**
num_colors = len(genre_pivot_relative.columns)
base_palette = sns.color_palette("husl", n_colors=num_colors)

# Adjust colors to be slightly **brighter & more saturated**
brighter_palette = [(mcolors.rgb_to_hsv(color)[0],
                     min(mcolors.rgb_to_hsv(color)[1] * 1.1, 1),
                     min(mcolors.rgb_to_hsv(color)[2] * 1.2, 1))
                    for color in base_palette]
brighter_palette = [mcolors.hsv_to_rgb(color) for color in brighter_palette]  # Convert back to RGB

# --- Stacked Area Chart to Show Genre Share Over Time ---
fig, ax = plt.subplots(figsize=(14, 7))  # Wider figure for better readability
genre_pivot_relative.plot(kind="area", stacked=True, alpha=0.85, color=brighter_palette, ax=ax)

plt.xlabel("Year")
plt.ylabel("Percentage of Total Box Office Revenue")
plt.title(f"Major Movie Genres Over Time (Relative Market Share, {roll}-Year Rolling Avg)")
plt.grid(axis="y", linestyle="--", linewidth=0.5)

# Fix legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, labels, loc="upper left", bbox_to_anchor=(1, 1), fontsize=8, title="Genre", ncol=2)

plt.show()
```



Percent of Total Box Office Revenue From Top 5 Movies Over Time

```
import plotly.express as px

# Load dataset
df = pd.read_csv("movies.csv", on_bad_lines="skip", nrows=40000)

# Convert release_date to datetime and extract year
df["release_date"] = pd.to_datetime(df["release_date"], errors="coerce")
df["release_year"] = df["release_date"].dt.year

# Convert revenue to numeric
df["revenue"] = pd.to_numeric(df["revenue"], errors="coerce")

# Drop missing values
df = df.dropna(subset=["revenue", "release_year"])

# Filter dataset to only include movies from 1940 onward
df_1940_present = df[df["release_year"] >= 1940]

# Compute total revenue per year
total_revenue_per_year = df_1940_present.groupby("release_year")["revenue"].sum()

# Identify the top 5 highest-grossing movies for each year
top_movies_per_year = df_1940_present.groupby("release_year").apply(lambda x: x.nlargest(5, "revenue")).reset_index(drop=True)

# Compute percentage of total revenue from top 5 movies
top_movies_revenue_per_year = top_movies_per_year.groupby("release_year")["revenue"].sum()
top_movies_percentage = (top_movies_revenue_per_year / total_revenue_per_year) * 100

# Convert to DataFrame for visualization
top_movies_percentage_df = top_movies_percentage.reset_index()
top_movies_percentage_df.columns = ["release_year", "percentage_of_total_revenue"]

# Create an interactive bar chart showing the percentage of revenue from the top 5 movies each year
fig = px.bar(
    top_movies_percentage_df,
    x="release_year",
    y="percentage_of_total_revenue",
    text="percentage_of_total_revenue",
    labels={"percentage_of_total_revenue": "Percentage of Total Revenue", "release_year": "Year"},
    title="Percentage of Total Box Office Revenue from Top 5 Movies (1940-Present)",
    color="percentage_of_total_revenue"
)

# Update layout for better readability
fig.update_traces(texttemplate="%{text:.1f}%", textposition="outside")
fig.update_layout(xaxis=dict(type="category"), yaxis_tickformat=".1f")

# Show the interactive plot
fig.show()
```

```
<ipython-input-40-0e843b6a6d24>:23: DeprecationWarning:

DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the
```

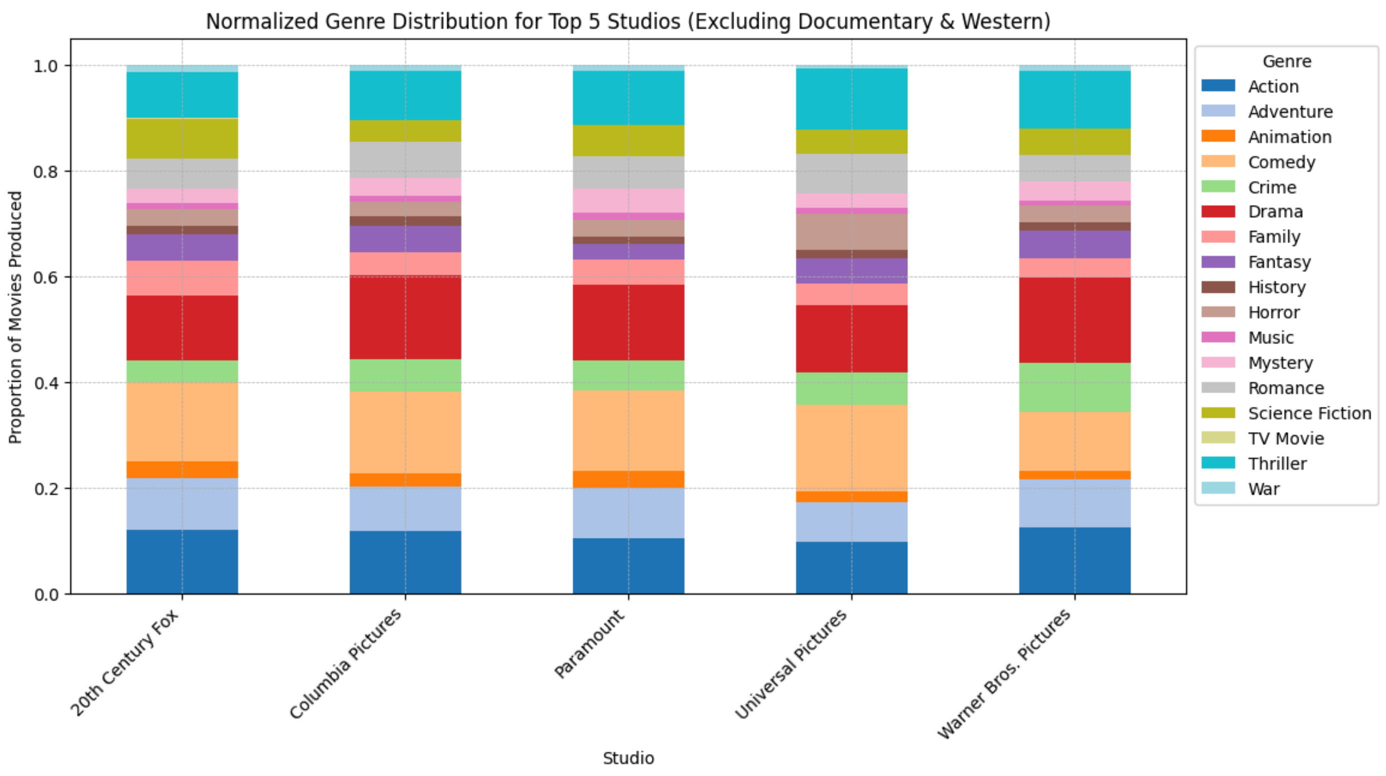


Genre Distribution for Top 5 Studios

```
movie_df_new.to_csv("movies1.csv", index=False)


# Load dataset
df = pd.read_csv("movies.csv", on_bad_lines="skip", nrows=10000)

# Drop missing values in key columns
df = df.dropna(subset=["genres", "production_companies"])

# Split genres and production companies into separate lists
df["genre_list"] = df["genres"].str.split(", ")
df["studio_list"] = df["production_companies"].str.split(", ")

# Explode both genres and studios into separate rows
df_exploded = df.explode("genre_list").explode("studio_list")

# Remove Documentary and Western genres
df_exploded = df_exploded[~df_exploded["genre_list"].isin(["Documentary", "Western"])]

# Identify top 5 studios by total movie count
top_studios = df_exploded["studio_list"].value_counts().nlargest(5).index.tolist()

# Filter dataset to only include movies from these top 5 studios
df_top_studios = df_exploded[df_exploded["studio_list"].isin(top_studios)]

# Compute movie count per genre for top 5 studios
studio_genre_distribution = df_top_studios.groupby(["studio_list", "genre_list"]).size().reset_index(name="movie_count")

# Pivot to have studios as rows and genres as columns
studio_pivot = studio_genre_distribution.pivot(index="studio_list", columns="genre_list", values="movie_count").fillna(0)

# Normalize so each studio's total movies sum to 1 (convert counts to proportions)
studio_pivot_normalized = studio_pivot.div(studio_pivot.sum(axis=1), axis=0)

# Generate unique colors for each genre
unique_genres = studio_pivot_normalized.columns
colors = plt.cm.get_cmap("tab20", len(unique_genres))  # Assign distinct colors
color_map = {genre: colors(i) for i, genre in enumerate(unique_genres)}

# --- Stacked Bar Chart: Top 5 Studios & Their Genre Distribution (Normalized) ---
```

```
studio_pivot_normalized.plot(kind="bar", stacked=True, figsize=(12, 6), color=[color_map[genre] for genre in studio_pivot_normal

plt.xlabel("Studio")
plt.ylabel("Proportion of Movies Produced")
plt.title("Normalized Genre Distribution for Top 5 Studios (Excluding Documentary & Western)")
plt.xticks(rotation=45, ha="right")
plt.legend(title="Genre", bbox_to_anchor=(1, 1))
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()
```

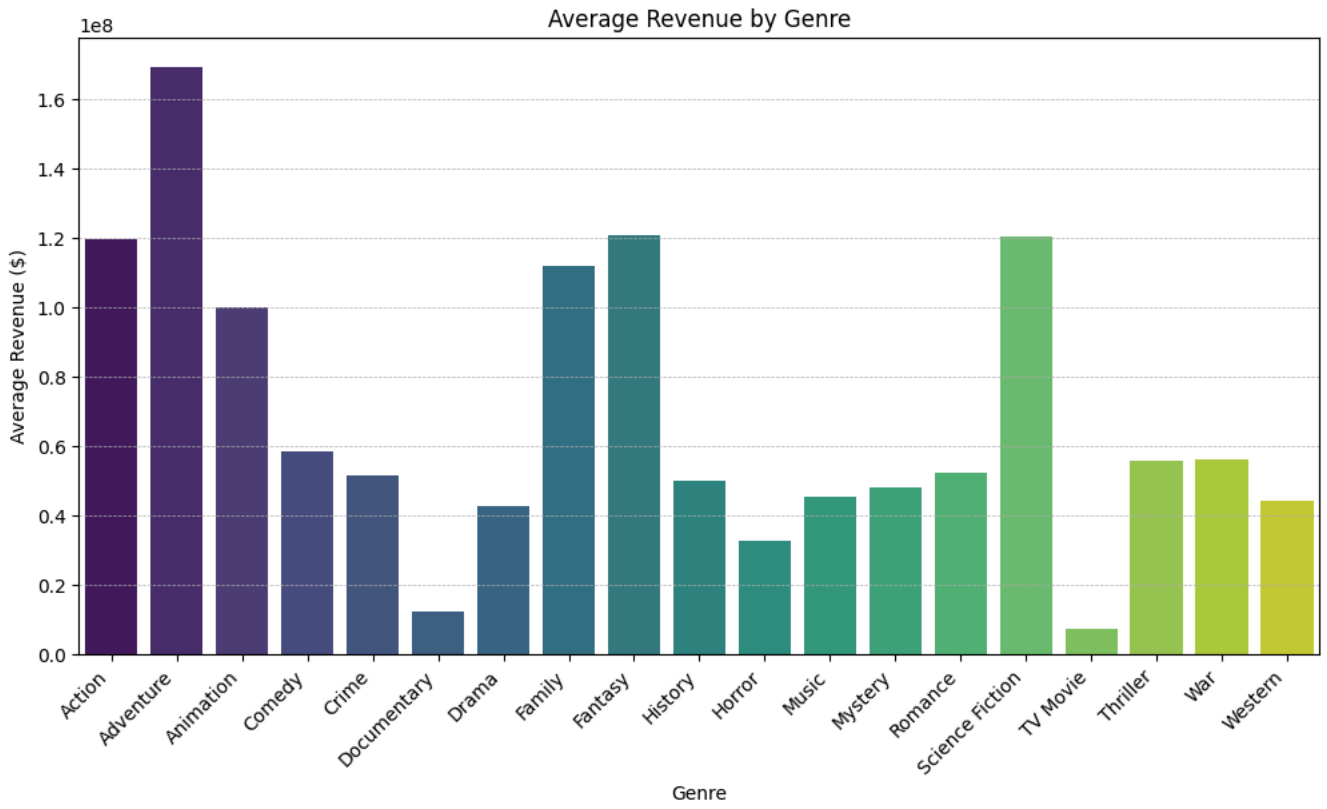
```
<ipython-input-42-43c450a1d203>:34: MatplotlibDeprecationWarning:

The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``
```



Average Revenue By Genre

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MultiLabelBinarizer

# Check if file exists before running
file_path = "movies.csv"  # Adjust this if needed
if not os.path.exists(file_path):
    print(f"Error: File '{file_path}' not found! Please check the file path.")
else:
    # Load dataset
    df = pd.read_csv(file_path, on_bad_lines="skip", nrows=10000)

    # Convert relevant columns to numeric
    df["revenue"] = pd.to_numeric(df["revenue"], errors="coerce")

    # Drop missing values in key columns
    df = df.dropna(subset=["vote_average", "revenue", "genres"])

    # One-Hot Encoding for Genres using MultiLabelBinarizer
    df["genre_list"] = df["genres"].apply(lambda x: x.split(", ") if isinstance(x, str) else ["Unknown"])
    mlb = MultiLabelBinarizer()
    genre_encoded = mlb.fit_transform(df["genre_list"])
    genre_df = pd.DataFrame(genre_encoded, columns=mlb.classes_)

    # Merge genre encodings back to original dataframe
    df = pd.concat([df, genre_df], axis=1)

    # Aggregate average vote score and revenue per genre
    genre_stats = pd.DataFrame()
    for genre in mlb.classes_:
        genre_stats.loc[genre, "avg_revenue"] = df[df[genre] == 1]["revenue"].mean()


    # Plot: Average Revenue by Genre
    plt.figure(figsize=(12, 6))
    sns.barplot(x=genre_stats.index, y=genre_stats["avg_revenue"], palette="viridis")
    plt.xlabel("Genre")
    plt.ylabel("Average Revenue ($)")
    plt.title("Average Revenue by Genre")
    plt.xticks(rotation=45, ha="right")
    plt.grid(axis="y", linestyle="--", linewidth=0.5)
    plt.show()
```

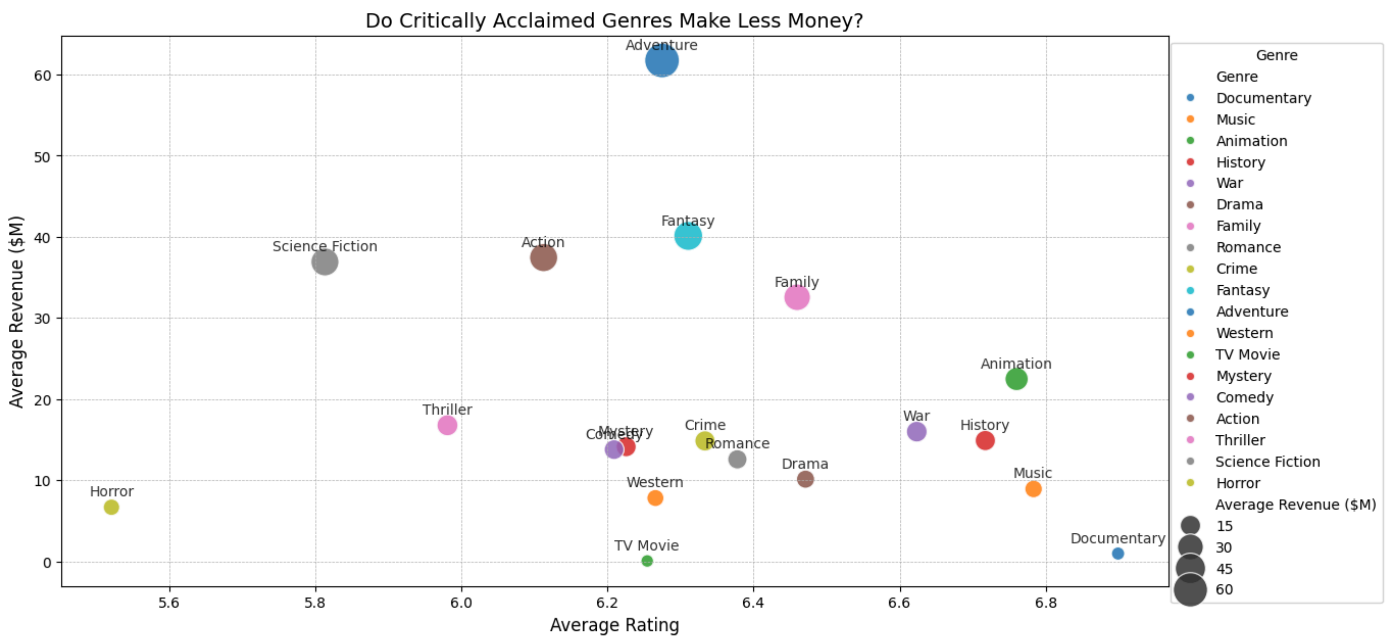
```
<ipython-input-50-6c4def3518c7>:38: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and
```



Do Critically Acclaimed Movies Also Make More Revenue?

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

# Load dataset
df = pd.read_csv("movies.csv", on_bad_lines="skip", nrows=50000)

# Convert release_date to datetime and extract year
df["release_date"] = pd.to_datetime(df["release_date"], errors="coerce")
df["Year"] = df["release_date"].dt.year
```

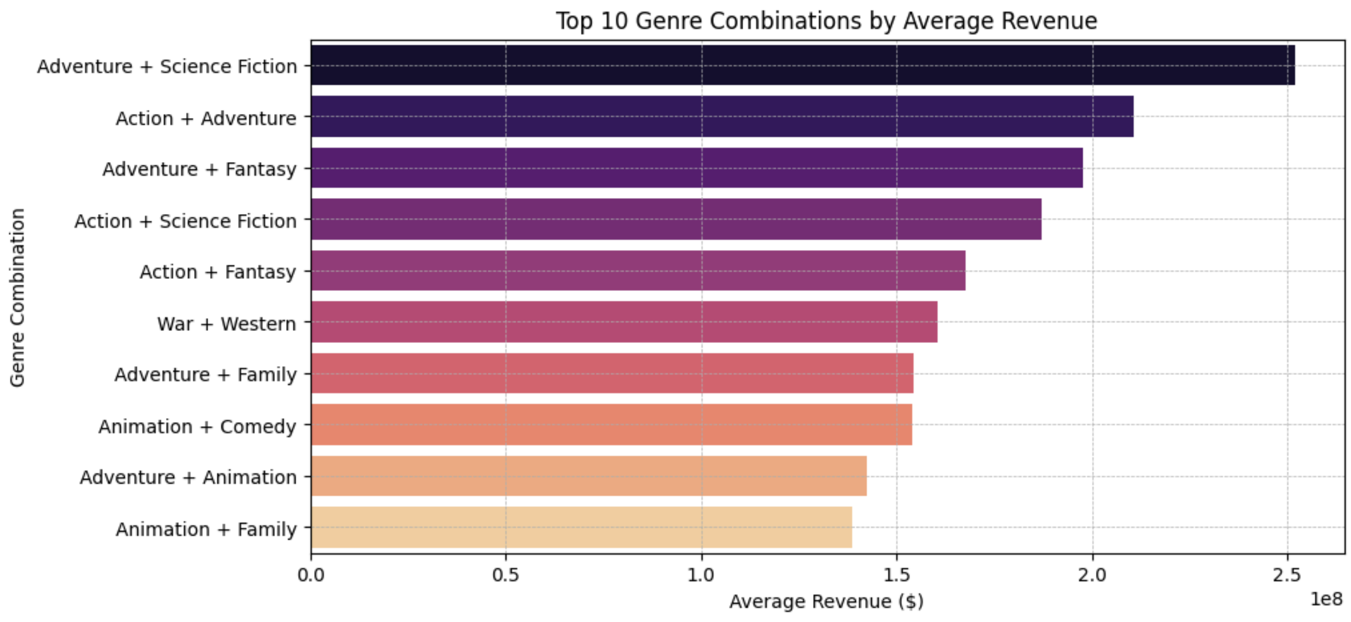
```
# Convert revenue and vote_average to numeric
df["BoxOffice"] = pd.to_numeric(df["revenue"], errors="coerce")
df["vote_average"] = pd.to_numeric(df["vote_average"], errors="coerce")

# Drop missing values
df = df.dropna(subset=["genres", "BoxOffice", "vote_average"])

# Split genres into separate rows
df_exploded = df.assign(Genre=df["genres"].str.split(", ")).explode("Genre")

# Compute average vote_average per genre
avg_rating_per_genre = df_exploded.groupby("Genre")["vote_average"].mean()

# Compute average revenue per genre
avg_revenue_per_genre = df_exploded.groupby("Genre")["BoxOffice"].mean()

# Combine both into a single dataframe for comparison
genre_analysis = pd.DataFrame({
    "Average Rating": avg_rating_per_genre,
    "Average Revenue ($M)": avg_revenue_per_genre / 1_000_000  # Convert to millions
}).reset_index()

# Sort by average rating
genre_analysis = genre_analysis.sort_values(by="Average Rating", ascending=False)

# --- 📊 Scatter Plot: Ratings vs Revenue ---
plt.figure(figsize=(14, 7))

# Use a colorblind-friendly palette
palette = sns.color_palette("tab10", len(genre_analysis))

scatter = sns.scatterplot(
    data=genre_analysis,
    x="Average Rating",
    y="Average Revenue ($M)",
    hue="Genre",
    size="Average Revenue ($M)",
    sizes=(80, 600),  # Adjust bubble sizes
    alpha=0.85,
    palette=palette
)

# Add labels slightly offset from points (avoiding overlap)
for i, row in genre_analysis.iterrows():
    plt.text(row["Average Rating"], row["Average Revenue ($M)"] + 1,
             row["Genre"], fontsize=10, ha='center', va='bottom', alpha=0.8)

# Labels
plt.xlabel("Average Rating", fontsize=12)
plt.ylabel("Average Revenue ($M)", fontsize=12)
plt.title("Do Critically Acclaimed Genres Make Less Money?", fontsize=14)

# Adjust legend
plt.legend(loc="upper right", bbox_to_anchor=(1.2, 1), title="Genre", fontsize=10)

plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()
```



Most Successful Genre Combinations by Average revenue

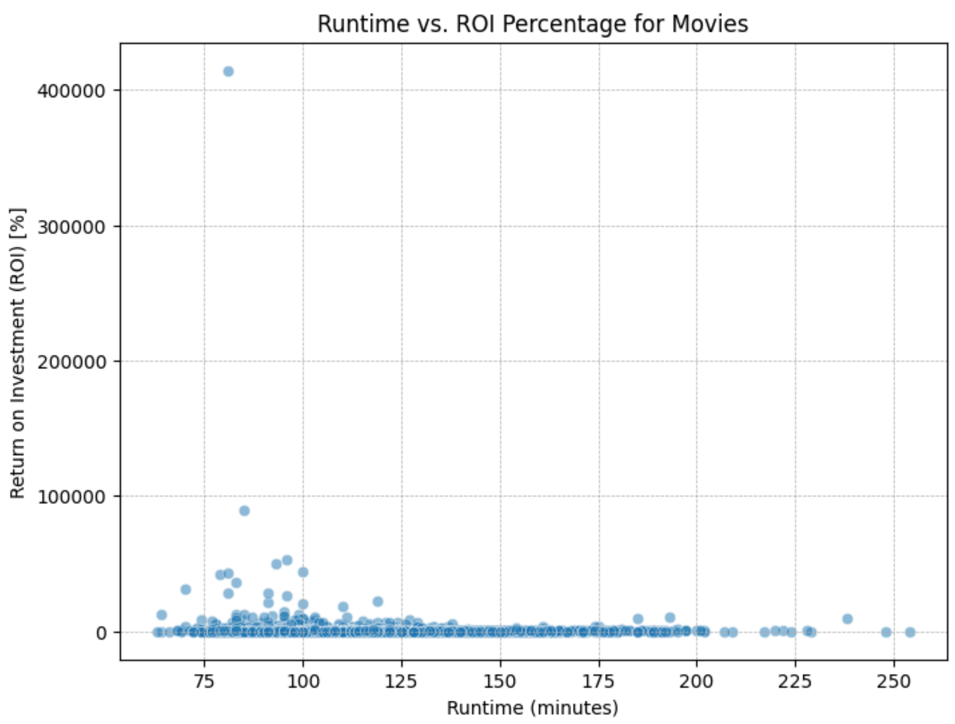
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import combinations

# Load dataset
df = pd.read_csv("movies.csv", on_bad_lines="skip", nrows=10000)

# Convert revenue to numeric
df["revenue"] = pd.to_numeric(df["revenue"], errors="coerce")

# Drop missing values in key columns
df = df.dropna(subset=["genres", "revenue"])

# Split genres into separate lists
df["genre_list"] = df["genres"].str.split(", ")

# Create genre pairs for each movie
df["genre_pairs"] = df["genre_list"].apply(lambda x: list(combinations(sorted(x), 2)) if len(x) > 1 else [])

# Explode genre pairs into separate rows
df_pairs = df.explode("genre_pairs").dropna(subset=["genre_pairs"])

# Convert tuples to string format for easy grouping
df_pairs["genre_pairs"] = df_pairs["genre_pairs"].apply(lambda x: " + ".join(x))

# Compute average revenue by genre pairing
genre_combo_revenue = df_pairs.groupby("genre_pairs")["revenue"].mean().reset_index()

# Get top 10 highest-grossing genre combinations
top_genre_combos = genre_combo_revenue.nlargest(10, "revenue")

# --- Bar Chart: Top 10 Genre Combinations by Revenue ---
plt.figure(figsize=(10, 5))
sns.barplot(data=top_genre_combos, x="revenue", y="genre_pairs", palette="magma")
plt.xlabel("Average Revenue ($)")
plt.ylabel("Genre Combination")
plt.title("Top 10 Genre Combinations by Average Revenue")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()
```

```
<ipython-input-47-5ea2bcb0a243>:35: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and
```

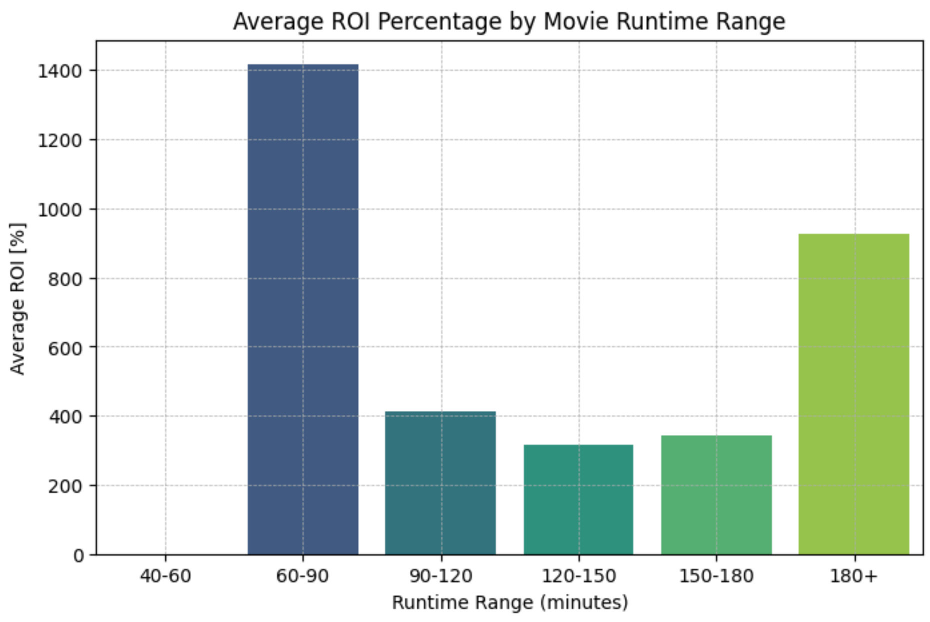


Do movies with higher budgets consistently make more money? Percent ROI by runtime

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

# Load dataset
df = pd.read_csv("movies.csv", on_bad_lines="skip", nrows=10000)

# Convert release_date to datetime
df["release_date"] = pd.to_datetime(df["release_date"], errors="coerce")

# Convert revenue, budget, and runtime to numeric
df["revenue"] = pd.to_numeric(df["revenue"], errors="coerce")
df["budget"] = pd.to_numeric(df["budget"], errors="coerce")
df["runtime"] = pd.to_numeric(df["runtime"], errors="coerce")

# Drop missing values in key columns
df = df.dropna(subset=["budget", "revenue", "runtime"])

# Filter out unrealistic values
df = df[(df["budget"] > 1000) & (df["revenue"] > 1000)]
df = df[(df["runtime"] >= 40) & (df["runtime"] <= 300)]

# Calculate ROI as a percentage
df["ROI"] = (df["revenue"] - df["budget"]) / df["budget"]
df["ROI_percentage"] = df["ROI"] * 100

# Drop extreme negative ROI values (where budget is high but revenue is near zero)
df = df[df["ROI"] > -1]

# --- Scatter Plot: Runtime vs ROI (Percentage) ---
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x="runtime", y="ROI_percentage", alpha=0.5)
plt.xlabel("Runtime (minutes)")
plt.ylabel("Return on Investment (ROI) [%]")
plt.title("Runtime vs. ROI Percentage for Movies")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()

# --- Average ROI Percentage by Runtime Range ---
df["runtime_bin"] = pd.cut(df["runtime"], bins=[40, 60, 90, 120, 150, 180, 300],
                           labels=["40-60", "60-90", "90-120", "120-150", "150-180", "180+"])
runtime_roi_percentage = df.groupby("runtime_bin")["ROI_percentage"].mean().reset_index()

# --- Bar Chart: Average ROI Percentage by Runtime Range ---
plt.figure(figsize=(8, 5))
sns.barplot(data=runtime_roi_percentage, x="runtime_bin", y="ROI_percentage", palette="viridis")
plt.xlabel("Runtime Range (minutes)")
plt.ylabel("Average ROI [%]")
plt.title("Average ROI Percentage by Movie Runtime Range")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.show()
```



```
<ipython-input-48-9e672b389053>:42: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False t

<ipython-input-48-9e672b389053>:46: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and
```



Lula's Visualizations

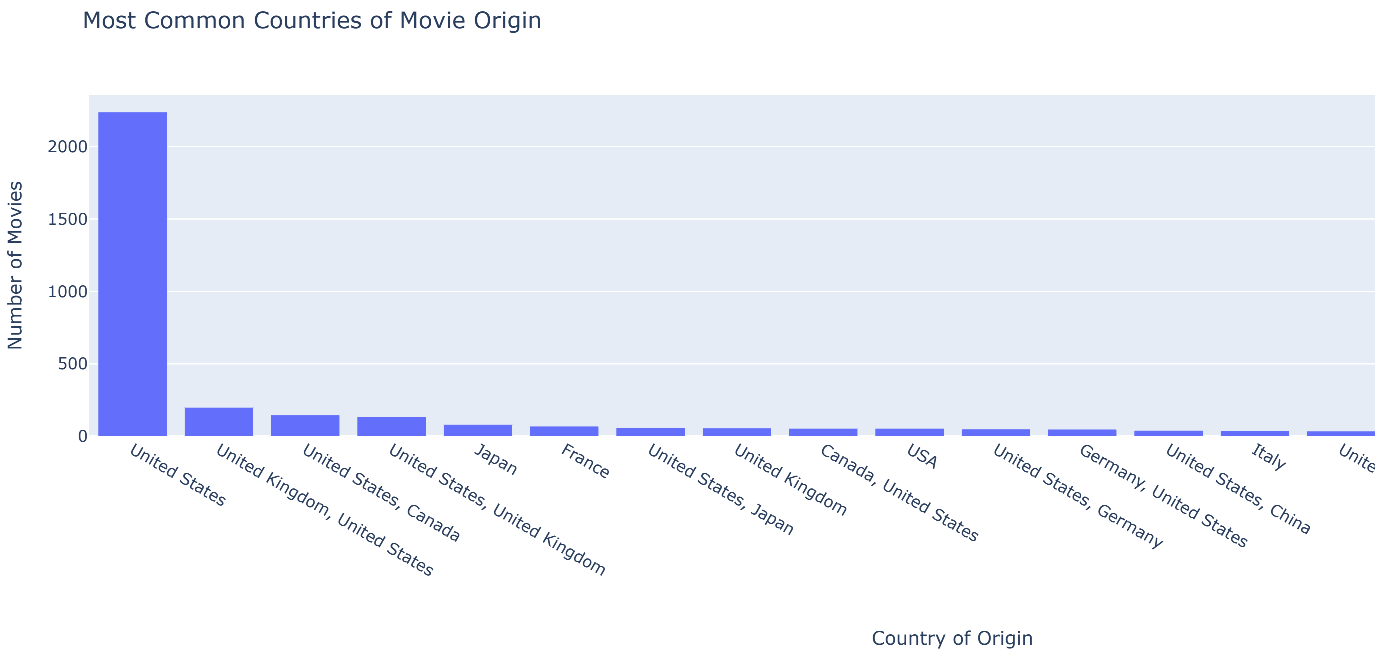
Extracting ratings from the ratings column

```
def extract_numeric(value):
    """ Convert rating values from different formats to a 0-100 scale. """
    if isinstance(value, str):
        if "/" in value:  # Handles '8.8/10' or '74/100'
            num, denom = value.split("/")
            return (float(num) / float(denom)) * 100
        elif "%" in value:  # Handles '87%'
            return float(value.strip('%'))
    return np.nan  # Return NaN for unexpected formats

# Explode the "Ratings" column (this assumes "Ratings" contains lists of dictionaries)
df_exploded = movie_df_new.explode("Ratings").reset_index(drop=True)

# Convert the "Ratings" column into a DataFrame (each dictionary becomes a column)
ratings_df = pd.json_normalize(df_exploded["Ratings"])

# Apply the function to convert ratings to a common 0-100 scale
ratings_df["Numeric Value"] = ratings_df["Value"].apply(extract_numeric)

# Drop the old "Ratings" column
df_exploded = df_exploded.drop(columns=["Ratings"])

# Merge the exploded DataFrame with the new ratings DataFrame
df_final = df_exploded.join(ratings_df)

# Pivot the ratings DataFrame to make each rater a column
df_pivoted = df_final.pivot_table(
    index=['Title'],  # Keep the movie title as the index
    columns='Source',  # Create columns for each rating source
    values='Numeric Value',  # Populate the cells with the numeric values of ratings
    aggfunc='first'  # If there are multiple ratings, take the first
).reset_index()

# Merge the pivoted ratings columns back into the original df_final
df_final_with_ratings = pd.merge(df_final, df_pivoted, on="Title", how="left")

# Show the final DataFrame with ratings from multiple sources as columns
df_final_with_ratings
```

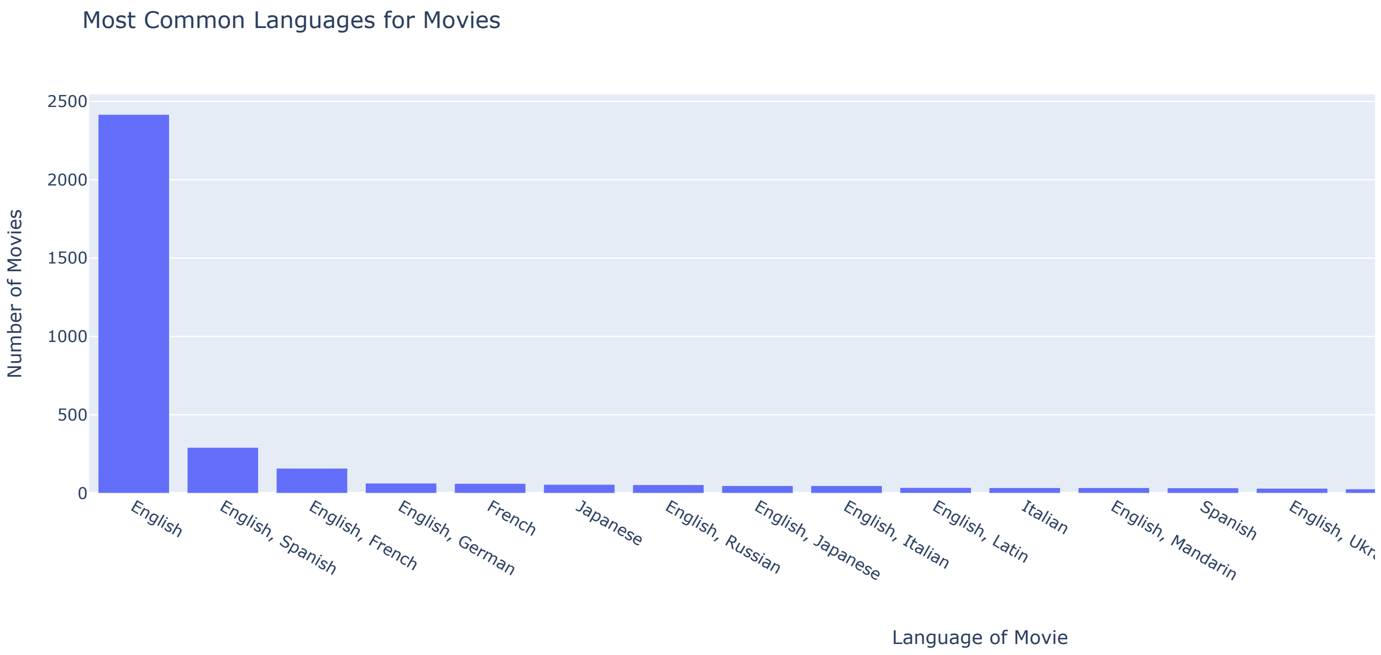
	Title	Year	Rated	Runtime	Genre	Director	Actors	Plot	Language	Country	...	Metascore	imdbRatin
0	Inception	2010	PG-13	148.0	Action	Christopher Nolan	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellio...	A thief who steals corporate secrets through t...	English, Japanese, French	United States, United Kingdom	...	74	8.
1	Inception	2010	PG-13	148.0	Action	Christopher Nolan	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellio...	A thief who steals corporate secrets through t...	English, Japanese, French	United States, United Kingdom	...	74	8.
2	Inception	2010	PG-13	148.0	Action	Christopher Nolan	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellio...	A thief who steals corporate secrets through t...	English, Japanese, French	United States, United Kingdom	...	74	8.
3	Interstellar	2014	PG-13	169.0	Adventure	Christopher Nolan	Matthew McConaughey, Anne Hathaway, Jessica Ch...	When Earth becomes uninhabitable in the future...	English	United States, United Kingdom, Canada	...	74	8.
4	Interstellar	2014	PG-13	169.0	Adventure	Christopher Nolan	Matthew McConaughey, Anne Hathaway, Jessica Ch...	When Earth becomes uninhabitable in the future...	English	United States, United Kingdom, Canada	...	74	8.
...	...	...	...	...	...	...	...	...	...	...	...	...	.
13875	Summer of 85	2020	Not Rated	101.0	Drama	François Ozon	Félix Lefebvre, Benjamin Voisin, Philippine Velge	When 16-year-old Alexis capsizes off the coast...	French, English	France, Belgium	...	65	6.
13876	Summer of 85	2020	Not Rated	101.0	Drama	François Ozon	Félix Lefebvre, Benjamin Voisin, Philippine Velge	When 16-year-old Alexis capsizes off the coast...	French, English	France, Belgium	...	65	6.
13877	Peter Rabbit 2: The Runaway	2021	PG	93.0	Animation	Will Gluck	Rose Byrne, Domhnall Gleeson, David Oyelowo	Thomas and Bea are now married and living with...	English, German	United States, Australia	...	43	6.
13878	Peter Rabbit 2: The Runaway	2021	PG	93.0	Animation	Will Gluck	Rose Byrne, Domhnall Gleeson, David Oyelowo	Thomas and Bea are now married and living with...	English, German	United States, Australia	...	43	6.
13879	Peter Rabbit 2: The Runaway	2021	PG	93.0	Animation	Will Gluck	Rose Byrne, Domhnall Gleeson, David Oyelowo	Thomas and Bea are now married and living with...	English, German	United States, Australia	...	43	6.

13880 rows × 21 columns

```
# Count occurrences of each country
country_counts = movie_df_new['Country'].value_counts().reset_index()
country_counts.columns = ['Country', 'Count']

fig = px.bar(country_counts.head(20), x='Country', y='Count',
             title='Most Common Countries of Movie Origin',
             labels={'Country': 'Country of Origin', 'Count': 'Number of Movies'})
fig.show()
```



```
#same thing but for languages
language_counts = movie_df_new['Language'].value_counts().reset_index()
language_counts.columns = ['Language', 'Count']

fig = px.bar(language_counts.head(20), x='Language', y='Count',
             title='Most Common Languages for Movies',
             labels={'Language': 'Language of Movie', 'Count': 'Number of Movies'})
fig.show()
```



```
boxoffice_awards = movie_df_new[['Title', 'BoxOffice', 'Awards']].copy()
boxoffice_awards
```

	Title	BoxOffice	Awards
0	Inception	$292,587,330	Won 4 Oscars. 159 wins & 220 nominations total
1	Interstellar	$203,227,580	Won 1 Oscar. 44 wins & 148 nominations total
2	The Dark Knight	$534,987,076	Won 2 Oscars. 164 wins & 165 nominations total
3	Avatar	$785,221,649	Won 3 Oscars. 91 wins & 131 nominations total
4	The Avengers	$623,357,910	Nominated for 1 Oscar. 39 wins & 81 nomination...
...	...	...	...
4909	The Leopard	N/A	Nominated for 1 Oscar. 10 wins & 6 nominations...
4910	Cassandra's Dream	$973,018	N/A
4911	The Royal Treatment	N/A	2 wins total
4912	Summer of 85	$71,788	6 wins & 27 nominations total
4913	Peter Rabbit 2: The Runaway	$40,501,717	2 wins & 6 nominations total

4914 rows × 3 columns

Next steps: Generate code with boxoffice_awards | View recommended plots | New interactive sheet

```python
import re
# Copy dataframe
boxoffice_awards = movie_df_new[['Title', 'BoxOffice', 'Awards']].copy()

# Convert BoxOffice to numeric
boxoffice_awards['BoxOffice'] = pd.to_numeric(
    boxoffice_awards['BoxOffice'].astype(str).str.replace('[$,]', '', regex=True),
    errors='coerce'
)

# Function to extract number of wins from Awards column
def extract_wins(awards):
    if isinstance(awards, str):
        match = re.search(r'(\d+) win', awards)
        if match:
            return int(match.group(1))
    return 0

boxoffice_awards['Wins'] = boxoffice_awards['Awards'].apply(extract_wins)

# Define properly matched box office bins and labels
bins = [0, 50_000_000, 200_000_000, 400_000_000, 650_000_000, float('inf')]
labels = ['0-50M', '50M-200M', '200M-400M', '400M-650M', '650M+']

# Categorize movies by box office revenue
boxoffice_awards['Box Office Category'] = pd.cut(boxoffice_awards['BoxOffice'], bins=bins, labels=labels)

# Group by box office category and calculate total wins & movie count
boxoffice_summary = boxoffice_awards.groupby('Box Office Category').agg(
    total_wins=('Wins', 'sum'),
    movie_count=('Title', 'count')
).reset_index()

# Compute average awards per movie for each category
boxoffice_summary['Avg Wins per Movie'] = boxoffice_summary['total_wins'] / boxoffice_summary['movie_count']

# Custom color palette for better differentiation
color_map = {
    '0-50M': 'crimson',
    '50M-200M': 'darkorange',
    '200M-400M': 'gold',
    '400M-650M': 'mediumseagreen',
    '650M+': 'royalblue'
}

# Create bar chart for average awards per movie
fig = px.bar(
    boxoffice_summary,
    x='Box Office Category',
    y='Avg Wins per Movie',
    title="Box Office Categories vs. Average Awards per Movie",
    labels={'Avg Wins per Movie': 'Average Awards per Movie'},
    labels={'Avg Wins per Movie': 'Average Awards per Movie'},
    category_orders={'Box Office Category': labels},
    color='Box Office Category',
    color_discrete_map=color_map,
    text=boxoffice_summary['Avg Wins per Movie'].round(2)  # Round to 2 decimal places
)

# Improve layout and readability
fig.update_traces(marker=dict(line=dict(width=1, color='black')))  # Add black outlines for contrast
fig.update_layout(
    title=dict(font=dict(size=18, family="Arial")),
    xaxis_title="Box Office Category",
    yaxis_title="Avg Awards per Movie",
    xaxis=dict(tickangle=-30),  # Rotate labels for spacing
    font=dict(size=12),
    showlegend=False  # No need for a legend since categories are on the x-axis
)

# Show the improved visualization
fig.show()
```

Machine Learning

Predicting Box Office From Several Parameters

```
def clean_year(year_str):
    if isinstance(year_str, str):
        numeric_part = re.sub(r'\D', '', year_str)
        if numeric_part:
            return int(numeric_part)
    return np.nan

movie_df_new['Year'] = movie_df_new['Year'].apply(clean_year)
```

```
from itertools import combinations
candidate_features = ['Runtime', 'Metascore', 'imdbRating', 'imdbVotes', 'Internet Movie Database', 'Metacritic', 'Rotten Tomatoe
for features in combinations(candidate_features, 4):
  print(features)


[list(features)]
```

```
('Runtime', 'Metascore', 'imdbRating', 'imdbVotes')
('Runtime', 'Metascore', 'imdbRating', 'Internet Movie Database')
('Runtime', 'Metascore', 'imdbRating', 'Metacritic')
('Runtime', 'Metascore', 'imdbRating', 'Rotten Tomatoes')
('Runtime', 'Metascore', 'imdbRating', 'Wins')
('Runtime', 'Metascore', 'imdbRating', 'Year')
('Runtime', 'Metascore', 'imdbVotes', 'Internet Movie Database')
('Runtime', 'Metascore', 'imdbVotes', 'Metacritic')
('Runtime', 'Metascore', 'imdbVotes', 'Rotten Tomatoes')
('Runtime', 'Metascore', 'imdbVotes', 'Wins')
('Runtime', 'Metascore', 'imdbVotes', 'Year')
('Runtime', 'Metascore', 'Internet Movie Database', 'Metacritic')
('Runtime', 'Metascore', 'Internet Movie Database', 'Rotten Tomatoes')
('Runtime', 'Metascore', 'Internet Movie Database', 'Wins')
('Runtime', 'Metascore', 'Internet Movie Database', 'Year')
('Runtime', 'Metascore', 'Metacritic', 'Rotten Tomatoes')
('Runtime', 'Metascore', 'Metacritic', 'Wins')
('Runtime', 'Metascore', 'Metacritic', 'Year')
('Runtime', 'Metascore', 'Rotten Tomatoes', 'Wins')
('Runtime', 'Metascore', 'Rotten Tomatoes', 'Year')
('Runtime', 'Metascore', 'Wins', 'Year')
('Runtime', 'imdbRating', 'imdbVotes', 'Internet Movie Database')
('Runtime', 'imdbRating', 'imdbVotes', 'Metacritic')
('Runtime', 'imdbRating', 'imdbVotes', 'Rotten Tomatoes')
('Runtime', 'imdbRating', 'imdbVotes', 'Wins')
('Runtime', 'imdbRating', 'imdbVotes', 'Year')
('Runtime', 'imdbRating', 'Internet Movie Database', 'Metacritic')
('Runtime', 'imdbRating', 'Internet Movie Database', 'Rotten Tomatoes')
('Runtime', 'imdbRating', 'Internet Movie Database', 'Wins')
('Runtime', 'imdbRating', 'Internet Movie Database', 'Year')
('Runtime', 'imdbRating', 'Metacritic', 'Rotten Tomatoes')
```

```
('Runtime', 'imdbRating', 'Metacritic', 'Wins')
('Runtime', 'imdbRating', 'Metacritic', 'Year')
('Runtime', 'imdbRating', 'Rotten Tomatoes', 'Wins')
('Runtime', 'imdbRating', 'Rotten Tomatoes', 'Year')
('Runtime', 'imdbRating', 'Wins', 'Year')
('Runtime', 'imdbVotes', 'Internet Movie Database', 'Metacritic')
('Runtime', 'imdbVotes', 'Internet Movie Database', 'Rotten Tomatoes')
('Runtime', 'imdbVotes', 'Internet Movie Database', 'Wins')
('Runtime', 'imdbVotes', 'Internet Movie Database', 'Year')
('Runtime', 'imdbVotes', 'Metacritic', 'Rotten Tomatoes')
('Runtime', 'imdbVotes', 'Metacritic', 'Wins')
('Runtime', 'imdbVotes', 'Metacritic', 'Year')
('Runtime', 'imdbVotes', 'Rotten Tomatoes', 'Wins')
('Runtime', 'imdbVotes', 'Rotten Tomatoes', 'Year')
('Runtime', 'imdbVotes', 'Wins', 'Year')
('Runtime', 'Internet Movie Database', 'Metacritic', 'Rotten Tomatoes')
('Runtime', 'Internet Movie Database', 'Metacritic', 'Wins')
('Runtime', 'Internet Movie Database', 'Metacritic', 'Year')
('Runtime', 'Internet Movie Database', 'Rotten Tomatoes', 'Wins')
('Runtime', 'Internet Movie Database', 'Rotten Tomatoes', 'Year')
('Runtime', 'Internet Movie Database', 'Wins', 'Year')
('Runtime', 'Metacritic', 'Rotten Tomatoes', 'Wins')
('Runtime', 'Metacritic', 'Rotten Tomatoes', 'Year')
('Runtime', 'Metacritic', 'Wins', 'Year')
('Runtime', 'Rotten Tomatoes', 'Wins', 'Year')
('Metascore', 'imdbRating', 'imdbVotes', 'Internet Movie Database')
('Metascore', 'imdbRating', 'imdbVotes', 'Metacritic')
```

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.impute import SimpleImputer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

# Features and Target
features = ['Title', 'imdbRating', 'Genre', 'Metacritic', 'Rotten Tomatoes']
X = df_final_with_ratings[features].copy()
y = df_final_with_ratings['BoxOffice'].copy()

# Convert 'N/A' or other non-numeric values to NaN
y = y.replace(['N/A', 'n/a', 'NA', '', None], np.nan)

# Ensure 'BoxOffice' is a string before replacing currency symbols
y = y.astype(str).str.replace('[\$,]', '', regex=True).str.replace(',', '').astype(float)  # Convert to float

# Convert numerical features to numeric (handle errors)
numeric_cols = ['imdbRating', 'Metacritic', 'Rotten Tomatoes']
X[numeric_cols] = X[numeric_cols].apply(pd.to_numeric, errors='coerce')

# Drop rows where BoxOffice is NaN (ensuring alignment)
data = pd.concat([X, y], axis=1).dropna(subset=['BoxOffice'])
X, y = data[features], data['BoxOffice']

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

# Define Transformers
vectorizer = TfidfVectorizer()
transformer = make_column_transformer(
    (make_pipeline(SimpleImputer(strategy='mean'), StandardScaler()), numeric_cols),  # Numerical pipeline
    (vectorizer, "Title"),  # Text processing
    (OneHotEncoder(handle_unknown='ignore', sparse_output=False), ["Genre"]),  # Categorical encoding
    remainder="drop"
)

# Build Pipeline with RandomForestRegressor
model = make_pipeline(transformer, RandomForestRegressor(n_estimators=100, random_state=42))

# Fit the model
model.fit(X_train, y_train)

# Predict
y_preds = model.predict(X_test)


from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
import numpy as np

# Calculate metrics
mae = mean_absolute_error(y_test, y_preds)
mse = mean_squared_error(y_test, y_preds)
```