Introduction

In today's data-driven entertainment industry, accurately predicting a movie's box office performance is crucial for production houses, distributors, and investors. Traditional prediction models mostly rely on static features such as cast, director, genre, budget, and historical box office trends. While these models offer baseline insights, they fall short in capturing real-time dynamics like audience excitement, social media buzz, and word-of-mouth sentiment — all of which significantly influence a movie's financial success.

Our project aims to bridge this gap by integrating sentiment and emotional signals derived from social media platforms such as Reddit, YouTube, and IMDB, alongside structured movie data. This enriched approach goes beyond just revenue forecasting — it empowers stakeholders with timely insights into public anticipation, emotional

resonance, and real-world audience perception prior to and during a movie's release cycle.

Problem statement

Traditional box office prediction models primarily rely on structured features such as cast, genre, budget, and past revenue performance. While useful, these models overlook a critical factor that increasingly influences a movie's commercial success: real-time public sentiment and emotional engagement across digital platforms. In today’s highly connected world, the success or failure of a film is significantly shaped by how audiences perceive and discuss it before and during its release. Public hype, emotional buzz, and online discourse play a vital role in shaping movie attendance and streaming behavior — yet current models fail to incorporate this dynamic, real-time social feedback loop.

Apporach

We propose that integrating sentiment polarity and emotional tone from public discussions will improve forecasting accuracy, helping stakeholders make smarter marketing, release, and investment decisions — ultimately minimizing financial risk and maximizing return.

Our project involves two types of data sources. The first is static or raw data, which we extract from IMDB. The second is real-time data from platforms like YouTube and Reddit, where we gather up-to-date public opinions about movies and generate sentiment scores. We use both types of data to train our models.

We trained three models using only structured metadata features: XGBoost, CatBoost, and LightGBM.

Sentiment & Emotion Signal Integration We extracted public sentiment and emotional reactions using pretrained machine learning models. Due to computational constraints, instead of fetching data for all movies directly from the Reddit and YouTube APIs, we used these pretrained models to generate sentiment and emotion scores.

Proposal Changes

Originally, the project included Twitter as a major sentiment source. However, due to Twitter's API restrictions and limited access to non-privileged developer data, we had to remove Twitter from the pipeline.

To compensate and strengthen our model, we introduced emotion analysis — going beyond sentiment polarity (positive/negative) to capture the type and depth of emotion

expressed by the audience. This enhancement helps differentiate between movies with similar sentiment but different emotional impact.

Team Contribution

This section outlines the individual contributions of each team member toward the successful execution of the project.

Task	Team Member
IMDB Data Collection	Nitish Kumar
Reddit Data Collection	Leonardo Ferreira
Reddit Data Analysis	Leonardo Ferreira
YouTube Data Collection	Aryan Shetty
YouTube Data Analysis	Aryan Shetty
Data Cleaning & Preprocessing	Sunil Kuruba
Exploratory Data Analysis (EDA)	Sunil Kuruba & Nitish Kumar
Machine Learning Model Development	Niharika Belavadi Shekar
Model Evaluation and Verification	Niharika Belavadi Shekar
Documentation	Sunil Kuruba

Expected Deliverables (Milestones)

Phase	Timeline	Deliverables
Phase 1	March (Week 1–3)	Data collection, cleaning, standardization
Phase 2	April (Week 1–2)	EDA, Sentiment analysis, feature engineering, model setup
Phase 3	April (Week 3–4)	Final model training, evaluation, and report writing

Reflection

We encountered some practical challenges with data extraction using the Twitter API, primarily due to restricted access to historical data. Additionally, we had to apply filters to both YouTube and Reddit datasets to align the extracted data with the release dates of the corresponding movies.

The project is progressing well—we estimate that around 80% of the work has been completed. Our hypotheses are strong, and the preliminary results look promising. At this stage, we are not facing any major roadblocks.

Overall, the teamwork has been exceptional. From collaborative coding and effective brainstorming sessions to detailed visualizations and smooth communication, our combined efforts have really come together to make significant progress.

IMDB Data Collection

Author: Nithish Kumar

Data Collection

1. Objective

The main goal was to compile a dataset of movies released between 2010 and 2020, including essential information such as movie titles, release dates, budgets, revenues, and additional metadata like cast, crew, genres, and production companies.

2. Source

All movie data was collected using **The Movie Database (TMDB)**, a free and reputable resource. Its API provides comprehensive metadata about movies, including box office figures, popularity scores, cast and crew information, and more.

3. Methodology

1. **Retrieving Movie IDs**
    - A call was made to TMDB's **Discover Movie** endpoint to retrieve a list of all movies released within the target timeframe (2010–2020).
    - The response contained basic movie attributes, including their unique IDs.
    - Results spanned multiple pages, so pagination was used to fetch all relevant movie IDs.
2. **Fetching Detailed Information**
    - For each movie ID obtained in the first step, a second API call was made to TMDB's **Movie Details** endpoint, using the parameter `append_to_response=credits` to fetch cast and crew data in a single request.
    - The response included fields such as budget, revenue, popularity score, genres, runtime, production companies, and full cast and crew credits.
3. **Data Storage**
    - After extracting the required information from each response, it was compiled into a structured format (e.g., a Python dictionary).

- The data was then stored in a **CSV file**, ensuring it could be easily analyzed or imported for EDA and training the model.

4. Outcome

By combining both steps (discovering movie IDs first, then pulling all details for each movie), a **comprehensive dataset** was created. This dataset covers a decade's worth of film releases (2010–2020) and includes all the necessary metadata for further analysis, such as financial figures, cast/crew breakdowns, and other relevant attributes.

5. Considerations

- **Rate Limits**: The TMDB API has usage limits, so a slight delay (e.g., `time.sleep()`) was introduced between requests to avoid hitting the rate limit.
- **Data Quality**: TMDB data quality depends on community and official contributions; hence, any missing fields (like budget or revenue) are a result of the source data not being available.
- **Pagination**: Proper pagination was handled to ensure that all movies are gathered, rather than just the first page of results.

In [1]:
```
import requests
import pandas as pd
import time
```

In [2]:
```
TMDB_API_KEY = "8faf313c846b2530bcfa8618eee4984a"
BASE_URL = 'https://api.themoviedb.org/3/'
```

In [3]:
```
# Function to get all Movies id's that are released in a particular year
def get_movies_by_year_range(start_year, end_year, page=1):
    url = BASE_URL+ "discover/movie"
    params = {
        'api_key': TMDB_API_KEY,
        'language': 'en-US',
        'sort_by': 'popularity.desc',
        'include_adult': 'false',
        'include_video': 'false',
        'page': page,
        'primary_release_date.gte': f'{start_year}-01-01',
        'primary_release_date.lte': f'{end_year}-12-31',
    }

    response = requests.get(url, params=params)
    return response.json()
```

In [ ]:
```
movie_ids = []

# Loop over each year from 2010 - 2020
for year in range(2010, 2020):
```

```
    print(f"\nFetching movies from {year}")
    for page in range(1, 501):  # Max 500 pages per year
        data = get_movies_by_year_range(year, year, page)
        if not data or 'results' not in data or len(data['results']) == 0:
            break

        ids_on_page = [movie['id'] for movie in data['results']]
        movie_ids.extend(ids_on_page)

        print(f"Year {year} - Page {page} -> Fetched {len(ids_on_page)} IDs"

        time.sleep(0.25)  # Respect TMDb rate limits (4 req/sec)

print(f"\nTotal movie IDs collected: {len(movie_ids)}")

# Save to CSV
movies_id_df = pd.DataFrame(movie_ids, columns=['movie_id'])
```

```
Fetching movies from 2010
Year 2010 - Page 1 -> Fetched 20 IDs
Year 2010 - Page 2 -> Fetched 20 IDs
Year 2010 - Page 3 -> Fetched 20 IDs
Year 2010 - Page 4 -> Fetched 20 IDs

Total movie IDs collected: 80
```

In [10]:
```
# Sample showing movies_id df for fewer pages
movies_id_df
```

Out[10]:

	movie_id
0	48650
1	27205
2	20526
3	38575
4	11324
...	...
75	65496
76	35552
77	50723
78	38199
79	10196

80 rows × 1 columns

In [ ]:
```
# Querying TMDB API to get all static info about a movie.
def get_tmdb_movie_data(tmdb_id):
    url = f"{BASE_URL}movie/{tmdb_id}?api_key={TMDB_API_KEY}&append_to_respo
```

```
    response = requests.get(url)

    if response.status_code != 200:
        print(f"Failed to get data for movie ID: {tmdb_id}")
        return None

    data = response.json()

    # Get director
    director = next((crew['name'] for crew in data['credits']['crew'] if cre

    # Get top 3 cast
    top_cast = [cast['name'] for cast in data['credits']['cast'][:3]]

    # Get keywords
    keywords = [kw['name'] for kw in data.get('keywords', {}).get('keywords'

    return {
        'movie_ID': data.get('id'),
        'IMDB_ID': data.get('imdb_id'),
        'title': data.get('title'),
        'vote_average': data.get('vote_average'),
        'vote_count': data.get('vote_count'),
        'status': data.get('status'),
        'Release Date': data.get('release_date'),
        'Budget': data.get('budget'),
        'Revenue': data.get('revenue'),
        'Popularity': data.get('popularity'),
        'Runtime': data.get('runtime'),
        'Language': data.get('original_language'),
        'Genres': ", ".join([genre['name'] for genre in data.get('genres', [
        'Production Companies': ", ".join([company['name'] for company in da
        'Director': director,
        'Top Cast': ", ".join(top_cast),
        'Keywords': ", ".join(keywords)
    }


results = []

for index, row in movies_id_df.iterrows():
    tmdb_id = row['movie_id']
    data = get_tmdb_movie_data(tmdb_id)
    if data:
        results.append(data)

# Create a DataFrame from results
enriched_df = pd.DataFrame(results)

# Save to CSV
enriched_df.to_csv("tmdb_enriched_movies.csv", mode='a', header=False, index

print("Data saved to tmdb_enriched_movies.csv")
enriched_df.head()
```

```
Data saved to tmdb_enriched_movies.csv
```

Out[ ]:

	movie_ID	IMDB_ID	title	vote_average	vote_count	status	Release Date	B
0	48650	tt1263750	Room in Rome	6.404	761	Released	2010-05-07	
1	27205	tt1375666	Inception	8.370	37319	Released	2010-07-15	1600(
2	20526	tt1104001	TRON: Legacy	6.495	7274	Released	2010-12-14	1700(
3	38575	tt1155076	The Karate Kid	6.543	6120	Released	2010-06-10	400(
4	11324	tt1130884	Shutter Island	8.200	24418	Released	2010-02-14	800(

In [ ]:

IMDB Data Preparation

Author: Sunil Kuruba, Nithish Kumar

IMDB Data Preparation & Cleaning

The IMDB dataset underwent a structured cleaning and preprocessing pipeline to ensure quality, consistency, and readiness for analysis. Below are the key steps involved:

1. Data Loading

- The original CSV file was loaded into a Pandas DataFrame for processing.

2. Column Cleaning

- Standardized column names:
  - Trimmed whitespace
  - Converted to lowercase
  - Replaced spaces with underscores (e.g., `Release Date` → `release_date`)

3. Data Type Conversion

- Converted key columns to appropriate types:
  - `release_date`: converted to `datetime`
  - `budget`, `revenue`, `runtime`, `vote_average`, `vote_count`: converted to numeric types

4. Handling Missing Values

- Dropped rows with missing or invalid `budget` or `revenue`
- Filled missing `runtime` values with the median

5. Exploding Multi-Value Fields

- Split and exploded comma-separated fields:
  - `genres` → `genres_list`
  - `top_cast` → `top_cast_list`
  - `keywords` → `keywords_list`

6. Text Normalization

- Cleaned string columns:

- Converted to lowercase
- Stripped leading/trailing whitespace

7. Duplicate Removal

- Removed duplicate rows based on `movie_id` and `title`

8. Feature Engineering

- Added `release_year` for temporal analysis
- Created exploded versions of multi-label fields for better grouping and filtering

9. Outlier Management

- Used visual techniques (e.g., box plots) to assess revenue skew and detect outliers

The cleaned dataset was used for generating insights such as revenue trends, genre popularity over time, and box office performance analysis. These transformations ensured the data was robust, analysis-ready, and suitable for visualization tasks.

In [ ]:
```
# Prerequisites
import pandas as pd
import numpy as np
import calendar
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.patches import Patch
from collections import Counter
```

In [182…
```
# Load CSV
df = pd.read_csv('tmdb_enriched_movies.csv')

# Clean column names
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
df.columns
```

Out[182…
```
Index(['movie_id', 'imdb_id', 'title', 'vote_average', 'vote_count', 'statu
s',
       'release_date', 'budget', 'revenue', 'popularity', 'runtime',
       'language', 'genres', 'production_companies', 'director', 'top_cas
t',
       'keywords'],
      dtype='object')
```

In [183…
```
df.head()
```

Out [183…

	movie_id	imdb_id	title	vote_average	vote_count	status	release_date
0	27205	tt1375666	Inception	8.369	37309	Released	2010-07-15
1	157336	tt0816692	Interstellar	8.453	36903	Released	2014-11-05
2	155	tt0468569	The Dark Knight	8.519	33688	Released	2008-07-16
3	19995	tt0499549	Avatar	7.588	32126	Released	2009-12-15
4	24428	tt0848228	The Avengers	7.735	31521	Released	2012-04-25

In [184…

```
# Convert data types
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
num_cols = ['budget', 'revenue', 'runtime', 'vote_average', 'vote_count']
for col in num_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

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

Out[184…

```
movie_id                  0
imdb_id                2786
title                     0
vote_average              0
vote_count                0
status                    0
release_date           1154
budget                    0
revenue                   0
popularity                0
runtime                   0
language                  0
genres                 1116
production_companies   2282
director               1208
top_cast                919
keywords               3111
dtype: int64
```

In [185…

```
# Handle missing values
df.dropna(subset=['budget', 'revenue'], inplace=True)

# Strip and lowercase text fields
text_cols = ['language', 'genres', 'production_companies', 'top_cast', 'keyw
for col in text_cols:
    df[col] = df[col].astype(str).str.strip().str.lower()
```

In [186…

```
# Split multi-value columns
df['genres_list'] = df['genres'].str.split(', ')
df['top_cast_list'] = df['top_cast'].str.split(', ')
df['keywords_list'] = df['keywords'].str.split(', ')

df[['genres', 'genres_list', 'top_cast', 'top_cast_list', 'keywords', 'keywo
```

Out [186…

	genres	genres_list	top_cast	top_cast_list	keywords	keywords_list
0	action, science fiction, adventure	[action, science fiction, adventure]	leonardo dicaprio, joseph gordon-levitt, ken w...	[leonardo dicaprio, joseph gordon-levitt, ken ...	rescue, mission, dreams, airplane, paris, fran...	[rescue, mission, dreams, airplane, paris, fra...
1	adventure, drama, science fiction	[adventure, drama, science fiction]	matthew mcconaughey, anne hathaway, michael caine	[matthew mcconaughey, anne hathaway, michael c...	rescue, future, spacecraft, race against time,...	[rescue, future, spacecraft, race against time...
2	drama, action, crime, thriller	[drama, action, crime, thriller]	christian bale, heath ledger, aaron eckhart	[christian bale, heath ledger, aaron eckhart]	joker, sadism, chaos, secret identity, crime f...	[joker, sadism, chaos, secret identity, crime ...
3	action, adventure, fantasy, science fiction	[action, adventure, fantasy, science fiction]	sam worthington, zoe saldaña, sigourney weaver	[sam worthington, zoe saldaña, sigourney weaver]	paraplegic, attachment to nature, culture clas...	[paraplegic, attachment to nature, culture cla...
4	science fiction, action, adventure	[science fiction, action, adventure]	robert downey jr., chris evans, mark ruffalo	[robert downey jr., chris evans, mark ruffalo]	new york city, superhero, shield, based on com...	[new york city, superhero, shield, based on co...

In [187…

```
# Remove duplicates
df.drop_duplicates(subset=['movie_id', 'title'], inplace=True)

# Remove invalid rows
df = df[(df['budget'] >= 0) & (df['revenue'] >= 0) & (df['runtime'] > 0)]

# Remove unwanted columns
df.drop(['imdb_id', 'status'], axis=1, inplace=True)

# Final preview
df.head()
```

Out[187…

	movie_id	title	vote_average	vote_count	release_date	budget	reve
0	27205	Inception	8.369	37309	2010-07-15	160000000	83903(
1	157336	Interstellar	8.453	36903	2014-11-05	165000000	746606
2	155	The Dark Knight	8.519	33688	2008-07-16	185000000	1004558
3	19995	Avatar	7.588	32126	2009-12-15	237000000	2923706
4	24428	The Avengers	7.735	31521	2012-04-25	220000000	151881(

In [188…

```
df.nlargest(3, 'budget')
```

Out[188…

	movie_id	title	vote_average	vote_count	release_date	budget	re
281	76600	Avatar: The Way of Water	7.609	12445	2022-12-14	460000000	23202
12005	1453985	Diggy and The Bandits	0.000	0	2001-09-11	447000000	
12007	1453767	Diggy and The Bandits	0.000	0	2001-09-11	447000000	

In [189…
```
df.nlargest(3, 'revenue')
```

Out[189…

	movie_id	title	vote_average	vote_count	release_date	budget	re
3	19995	Avatar	7.588	32126	2009-12-15	237000000	29237(
15	299534	Avengers: Endgame	8.238	26214	2019-04-24	356000000	27994
281	76600	Avatar: The Way of Water	7.609	12445	2022-12-14	460000000	23202

In [190…
```
df.nlargest(3, 'popularity')
```

Out[190…

	movie_id	title	vote_average	vote_count	release_date	budget	re
1094	7451	xXx	5.937	4394	2002-08-09	70000000	2774
1	157336	Interstellar	8.453	36903	2014-11-05	165000000	7466
53	238	The Godfather	8.686	21304	1972-03-14	6000000	245(

In [191…
```
df['genres'].unique()
```

Out[191…
```
array(['action, science fiction, adventure',
       'adventure, drama, science fiction',
       'drama, action, crime, thriller', ...,
       'animation, fantasy, romance', 'animation, fantasy, mystery',
       'action, comedy, mystery, horror'], shape=(2288,), dtype=object)
```

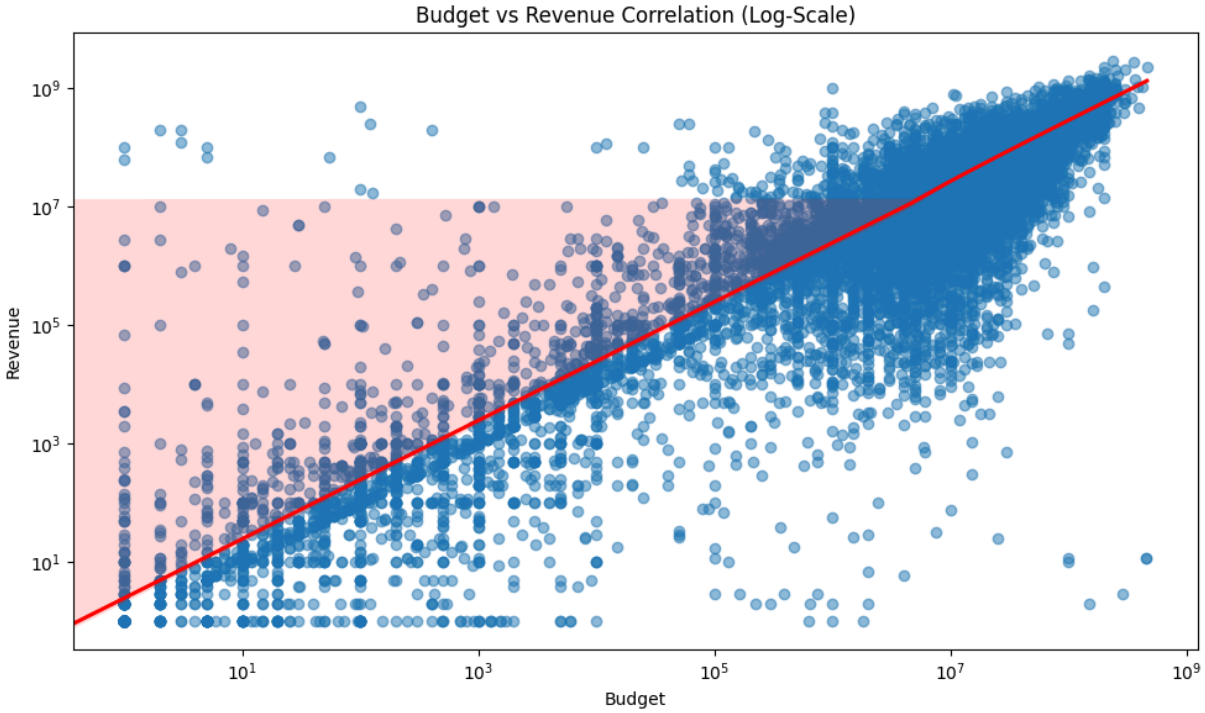
Exploratory Data Analysis

Visualization 1 - Budget VS Revenue

This scatter plot visualizes the relationship between a movie's **budget and revenue**, both plotted on a **logarithmic scale** to accommodate the wide range of values. Each blue dot represents an individual movie, and the red line denotes the **regression trend line**, showing the general direction of the relationship. The plot reveals a **positive correlation**—movies with higher budgets tend to earn higher revenues—but with substantial **variance and outliers**, especially among low-budget films. The pink-shaded area below the trend line highlights movies that **underperformed financially**, generating significantly less revenue than expected based on their budgets. This graph effectively illustrates how while high budgets often lead to higher revenue, success is not guaranteed, and many films fall below profitability expectations.

In [192…

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

sns.regplot(
    data=df,
    x='budget',
    y='revenue',
    scatter_kws={'alpha': 0.5},
    line_kws={'color': 'red'}
)

plt.xscale('log')
plt.yscale('log')
plt.title('Budget vs Revenue Correlation (Log-Scale)')
plt.xlabel('Budget')
plt.ylabel('Revenue')

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



Visualization 2 - Top 10 Genres by Average Revenue

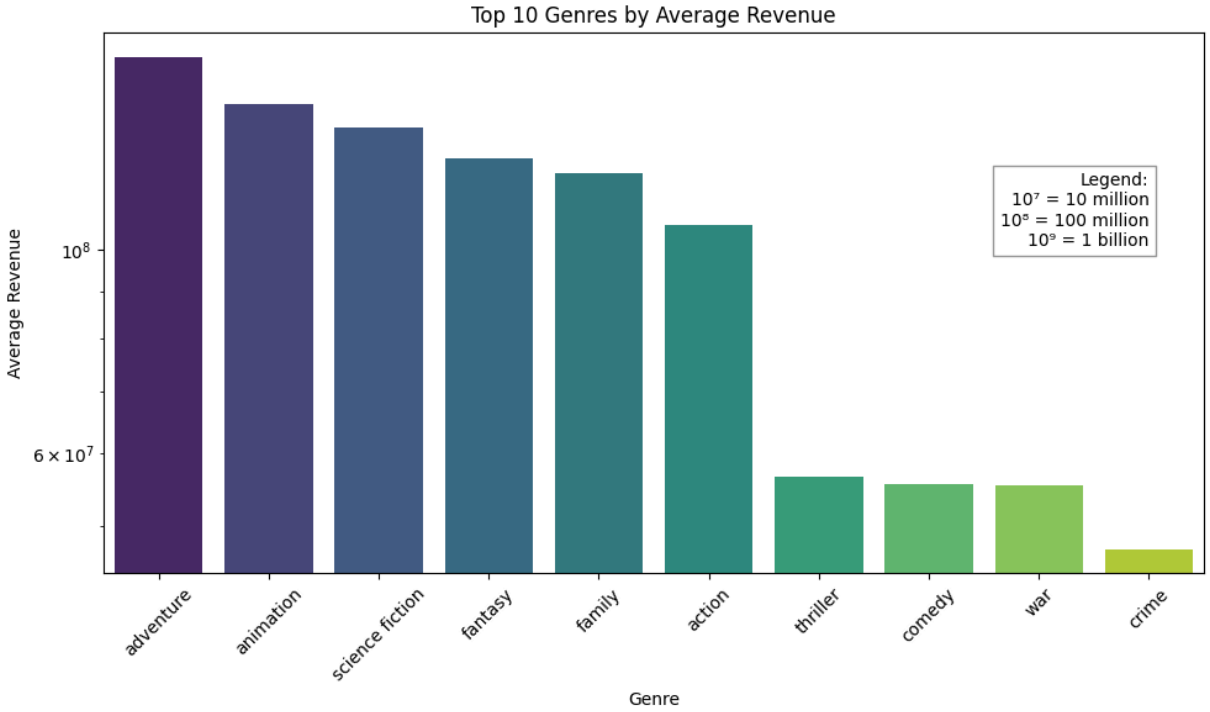
This bar chart displays the **top 10 movie genres ranked by their average revenue**, providing insight into which genres tend to generate the most box office success. The Y-axis uses a **logarithmic scale**, and a legend is included to help interpret values such as (10^7 = 10) million, (10^8 = 100) million, and (10^9 = 1) billion dollars. Among the genres, **Adventure** leads with the highest average revenue, followed closely by **Animation**, **Science Fiction**, and **Fantasy**, indicating strong commercial performance for visually immersive and franchise-driven content. Genres like **Thriller**, **Comedy**, **War**, and **Crime** appear on the lower end of the chart, suggesting they generally yield lower average returns. This graph effectively highlights genre profitability, reinforcing that high-budget, high-concept genres dominate the global box office.

Distribution of Movies by Genre (Pie Chart). This pie chart visualizes the proportional distribution of movies across genres. The most represented genres include: Drama (23%): Dominates the dataset, indicating a strong industry preference or audience interest. Comedy (16.8%) and Thriller (11.1%): Also highly represented, reflecting their broad appeal. Other notable genres: Action (11%), Romance (8.7%), Adventure (7.4%), and Crime (7.1%). Smaller slices such as Family, Science Fiction, and Horror indicate niche or secondary genres in terms of production volume. This visualization provides a quick overview of genre popularity based on movie count

In [193…

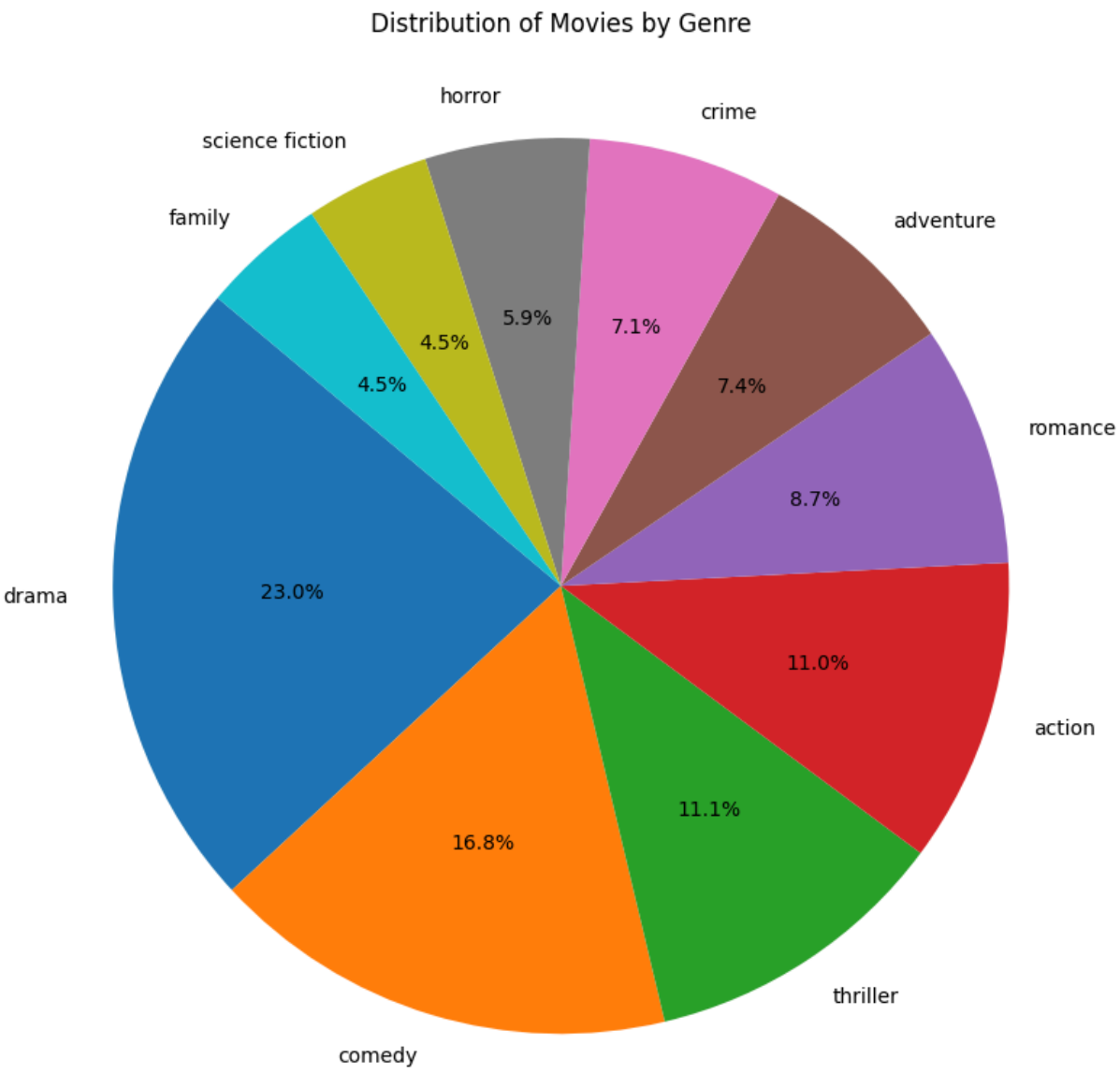
```
# Explode genre lists
genre_revenue = df.explode('genres_list')

# Compute average revenue per genre
genre_avg_revenue = genre_revenue.groupby('genres_list')['revenue'].mean().s
```

```
# Reset index for Seaborn
genre_avg_revenue = genre_avg_revenue.reset_index()

# Plot using seaborn
plt.figure(figsize=(10, 6))
sns.barplot(
    data=genre_avg_revenue,
    x='genres_list',
    y='revenue',
    hue='genres_list',
    palette='viridis',
    legend=False
)

# Customize plot
plt.yscale('log')
plt.title('Top 10 Genres by Average Revenue')
plt.ylabel('Average Revenue')
plt.xlabel('Genre')
plt.xticks(rotation=45)

# Add explanatory legend box
plt.text(
    x=9, y=1e8,
    s='Legend:\n10⁷ = 10 million\n10⁸ = 100 million\n10⁹ = 1 billion',
    fontsize=10,
    ha='right',
    va='bottom',
    bbox=dict(facecolor='white', alpha=0.8, edgecolor='gray')
)

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



In [194…

```
df_exploded = df.explode('genres_list')
genre_counts = df_exploded['genres_list'].value_counts().head(10)  # top 10

# Plot pie chart
plt.figure(figsize=(8, 8))
plt.pie(genre_counts, labels=genre_counts.index, autopct='%1.1f%%', startang
plt.title('Distribution of Movies by Genre')
plt.axis('equal')  # ensures pie is a circle
plt.tight_layout()
plt.show()
```

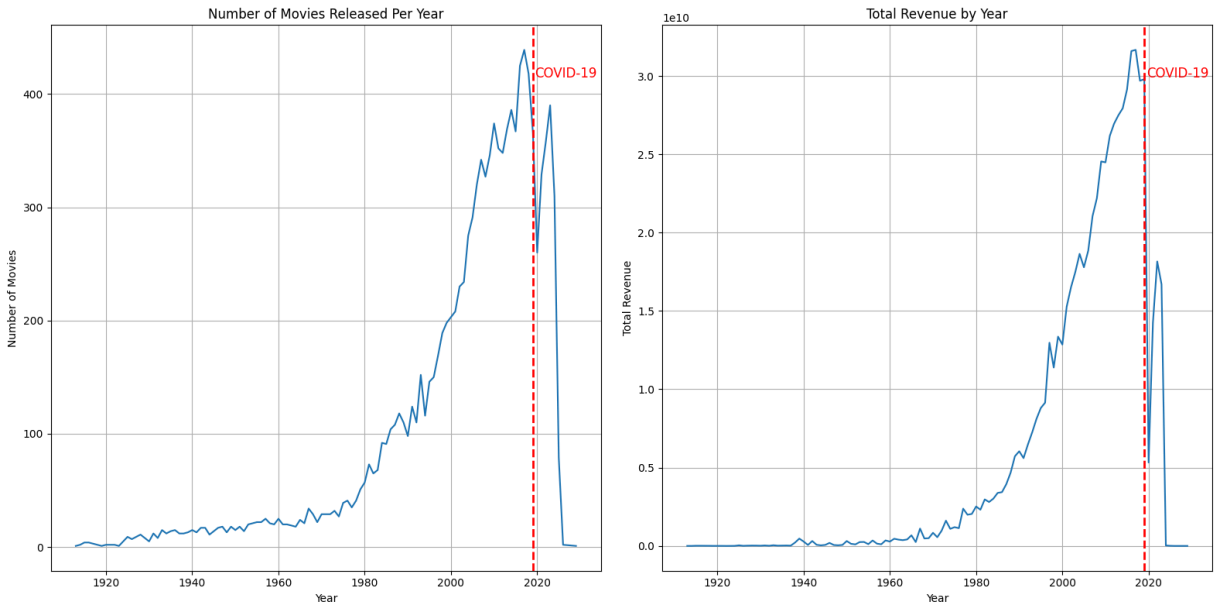


Visualization 3 - Number of Movies Released and Total Revenue Per year

These side-by-side line plots depict the impact of the COVID-19 pandemic on the global film industry. The **left plot** shows the number of movies released each year, while the **right plot** displays the total revenue generated annually. Both charts demonstrate a steady growth trajectory from the mid-20th century, peaking around **2018–2019**. However, the vertical red dashed line at **2019**, annotated with **"COVID-19"**, highlights a pivotal moment. Post-2019, both metrics exhibit a **dramatic decline**, reflecting the widespread disruptions caused by the pandemic, including halted productions, theater closures, and reduced box office activity. These visuals clearly illustrate how COVID-19 triggered a major downturn in movie releases and earnings after decades of consistent industry growth.

In [195…

```
df['release_year'] = df['release_date'].dt.year
movies_per_year = df['release_year'].value_counts().sort_index()
revenue_per_year = df.groupby('release_year')['revenue'].sum().sort_index()

# Create subplots
fig, axes = plt.subplots(ncols=2, figsize=(16, 8))

# Plot 1: Number of Movies Released Per Year
sns.lineplot(x=movies_per_year.index, y=movies_per_year.values, ax=axes[0])
axes[0].set_title('Number of Movies Released Per Year')
axes[0].set_xlabel('Year')
axes[0].set_ylabel('Number of Movies')
axes[0].axvline(x=2019, color='red', linestyle='--', linewidth=2)
axes[0].text(2019 + 0.5, axes[0].get_ylim()[1]*0.9, 'COVID-19', color='red',
axes[0].grid(True)

# Plot 2: Total Revenue by Year
sns.lineplot(x=revenue_per_year.index, y=revenue_per_year.values, ax=axes[1]
axes[1].set_title('Total Revenue by Year')
axes[1].set_xlabel('Year')
axes[1].set_ylabel('Total Revenue')
axes[1].axvline(x=2019, color='red', linestyle='--', linewidth=2)
axes[1].text(2019 + 0.5, axes[1].get_ylim()[1]*0.9, 'COVID-19', color='red',
axes[1].grid(True)

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



Visualization 4 - Revenue Distribution by Genre

This box plot visualizes the **distribution of movie revenues** across six popular genres: Action, Adventure, Drama, Thriller, Comedy, and Romance. The Y-axis is on a **logarithmic scale**, capturing a wide range of revenue values and helping to identify patterns and outliers. Each box shows the **interquartile range (IQR)**, with the horizontal line inside representing the **median revenue**. Notably, **Adventure and Action** genres

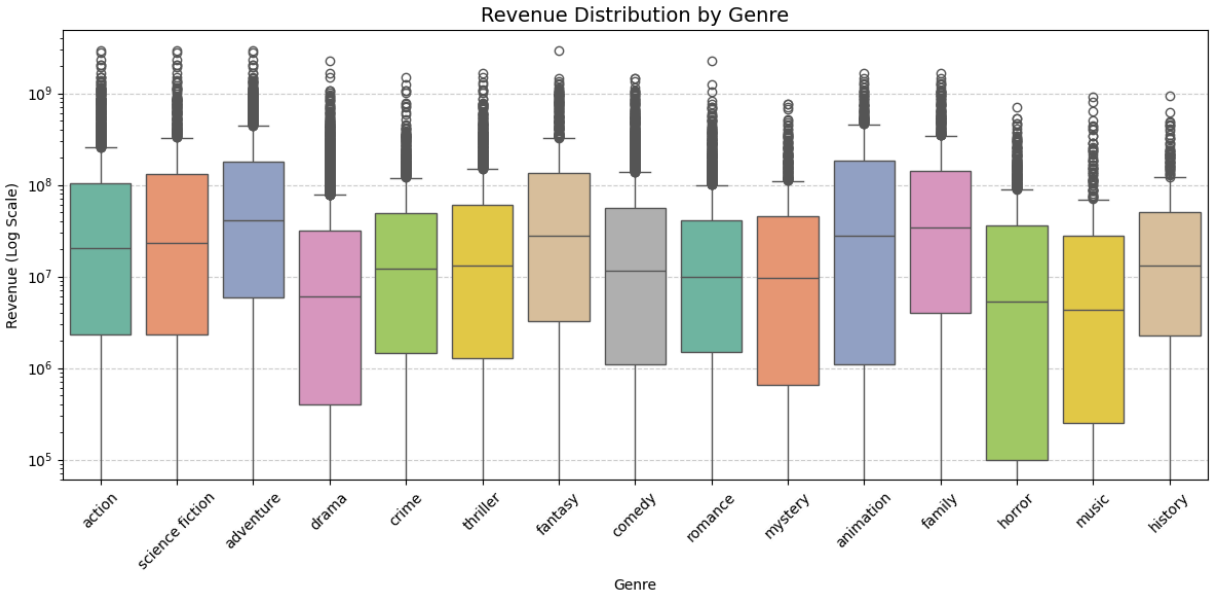
exhibit higher median revenues compared to others, suggesting stronger box office performance on average. All genres display a substantial number of **outliers**, indicating that while most films earn within a certain range, a few achieve exceptional financial success. This visualization highlights both the central tendencies and the variability in revenue performance across different movie genres.

Average Profit per Genre per Year (Heatmap)

This heatmap displays the average profit earned by movies in each genre over time (2000–2022). Each cell's color intensity represents the average profit, with darker blues indicating higher earnings. Adventure and Fantasy genres consistently show high profitability across multiple years. Family movies have performed especially well in several recent years, reflecting their enduring audience appeal. Comedy, Drama, Horror, and Crime show relatively lower profits on average, despite having high production counts (as seen in the pie chart). This heatmap is useful for understanding genre profitability trends over time and identifying which genres maintain strong financial performance year after year.

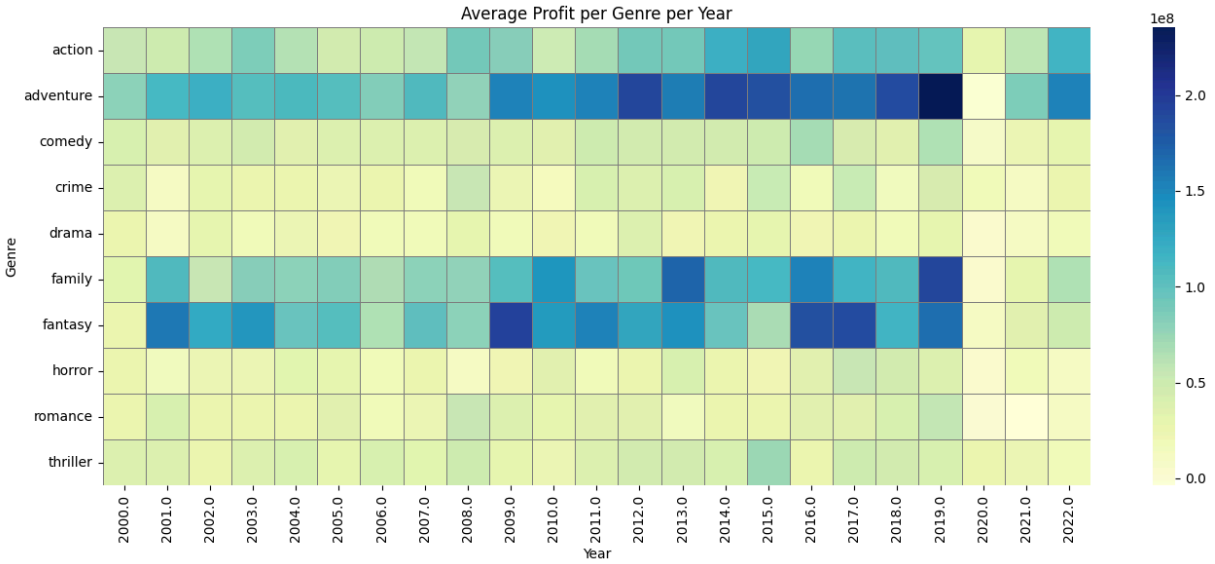
In [208…

```
# Filter for top 6 popular genres
popular_genres = df.explode('genres_list')
popular_genres = popular_genres[popular_genres['genres_list'].isin(
    popular_genres['genres_list'].value_counts().head(15).index)]

plt.figure(figsize=(12, 6))
sns.boxplot(
    x='genres_list',
    y='revenue',
    data=popular_genres,
    hue='genres_list',
    palette='Set2',
    legend=False
)


plt.yscale('log')
plt.title('Revenue Distribution by Genre', fontsize=14)
plt.xlabel('Genre')
plt.ylabel('Revenue (Log Scale)')
plt.xticks(rotation=45)
plt.grid(True, axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



In [197…

```
df['release_year'] = df['release_date'].dt.year
df['profit'] = df['revenue'] - df['budget']
df_exploded = df.explode('genres_list')

df_exploded = df_exploded[df_exploded['release_year'].between(2000, 2022)]
top_genres = df_exploded['genres_list'].value_counts().head(10).index
df_filtered = df_exploded[df_exploded['genres_list'].isin(top_genres)]

# Create pivot table with average profit
pivot_table = df_filtered.pivot_table(
    index='genres_list',
    columns='release_year',
    values='profit',
    aggfunc='mean'
)

# Plot heatmap
plt.figure(figsize=(14, 6))
sns.heatmap(pivot_table, cmap='YlGnBu', linewidths=0.5, linecolor='gray', ar
plt.title('Average Profit per Genre per Year')
plt.xlabel('Year')
plt.ylabel('Genre')
plt.tight_layout()
plt.show()
```

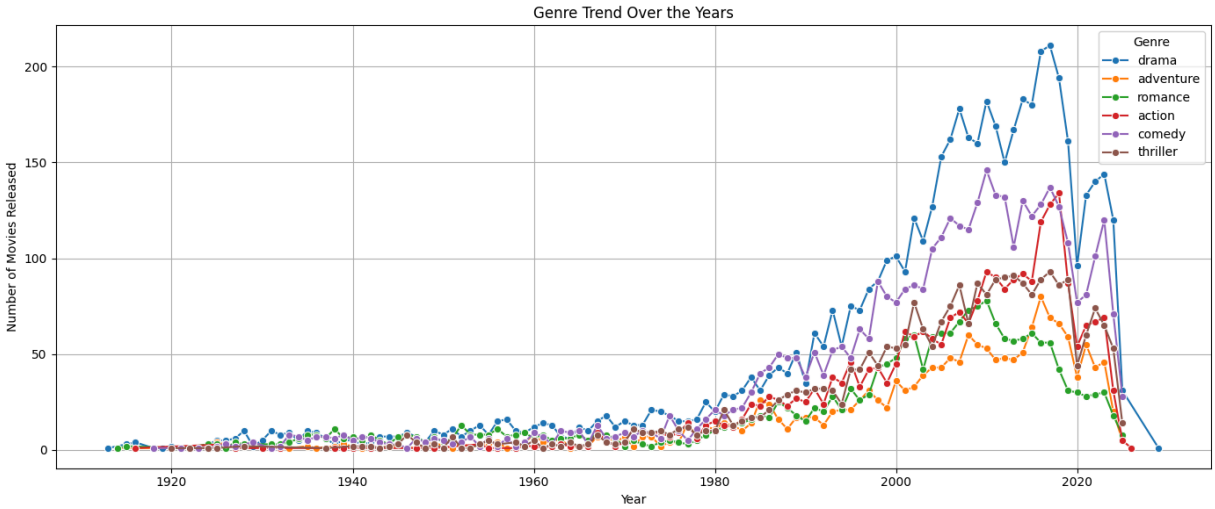


Visualization 5 - Genre Trend Over the Years

From the plot, we observe that Drama has consistently been the most released genre, peaking around 2015–2019 with over 200 films per year. Comedy and Thriller also show strong upward trends, especially from the 1980s onward. Action and Adventure genres saw significant growth in the 2000s and 2010s, aligning with the rise of blockbuster cinema. Romance maintained steady popularity but started declining in the late 2010s. There is a sharp drop in all genres after 2019, clearly indicating the impact of the COVID-19 pandemic on film production globally

In [198…

```
df['release_year'] = df['release_date'].dt.year
df_exploded = df.explode('genres_list')

# Filter to include only popular genres (optional)
top_genres = df_exploded['genres_list'].value_counts().head(6).index
df_filtered = df_exploded[df_exploded['genres_list'].isin(top_genres)]

# Group by year and genre to count number of movies
genre_trend = df_filtered.groupby(['release_year', 'genres_list']).size().re

# Plot the trend
plt.figure(figsize=(14, 6))
sns.lineplot(data=genre_trend, x='release_year', y='movie_count', hue='genre

plt.title('Genre Trend Over the Years')
plt.xlabel('Year')
plt.ylabel('Number of Movies Released')
plt.grid(True)
plt.legend(title='Genre')
plt.tight_layout()
plt.show()
```



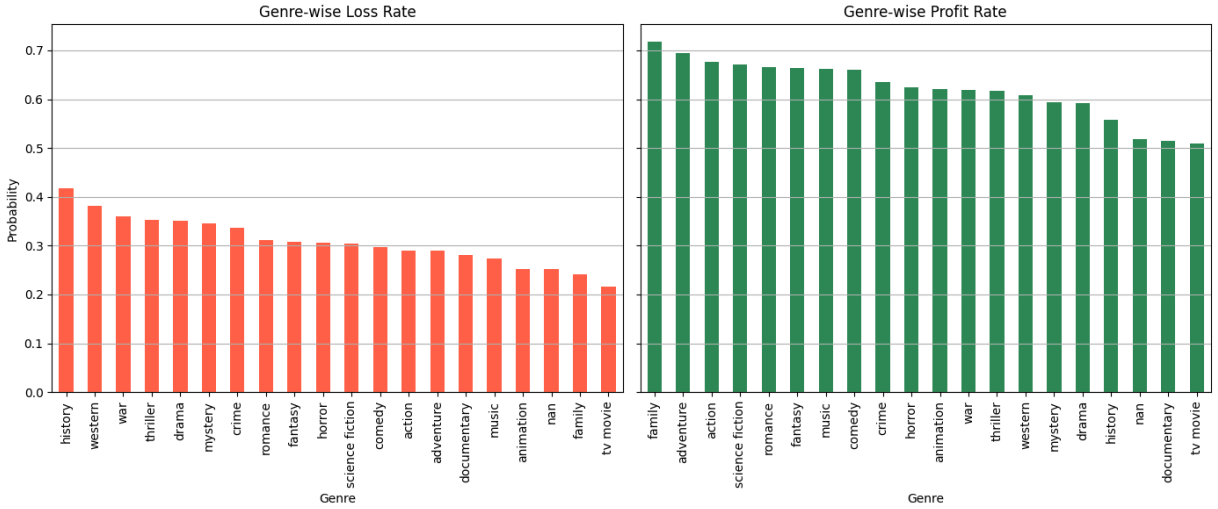
Visualization 6 - Profit vs Loss by Genre

This graph presents a comparative view of **genre-wise loss and profit rates** in the film industry. The **left plot** illustrates the proportion of movies that incurred a **loss** in each genre, with *History*, *Western*, and *War* genres topping the list—indicating that a significant share of films in these categories failed to recover their budget. On the **right plot**, we see the **profit rate** for each genre, with *Family*, *Adventure*, and *Action* films showing the highest probability of turning a profit. Interestingly, genres like *Documentary*, *TV Movie*, and *History* consistently appear at the lower end of the profit spectrum and higher end of the loss spectrum, suggesting these may be riskier investments from a financial perspective. This side-by-side visualization clearly demonstrates how profitability varies by genre and can guide production or investment decisions.

In [199…

```
# Calculate profit
df['profit'] = df['revenue'] - df['budget']

# Explode genres into individual rows
df_exploded = df.explode('genres_list')

# Create loss and profit flags
df_exploded['is_loss'] = df_exploded['profit'] < 0
df_exploded['is_profit'] = df_exploded['profit'] > 0

# Calculate percentage loss and profit per genre
loss_rate = df_exploded.groupby('genres_list')['is_loss'].mean()
profit_rate = df_exploded.groupby('genres_list')['is_profit'].mean()

# Combine both into one DataFrame
rate_df = pd.DataFrame({
    'Loss Rate': loss_rate,
    'Profit Rate': profit_rate
})

rate_df_loss = rate_df.sort_values('Loss Rate', ascending=False)
```

```
rate_df_profit = rate_df.sort_values('Profit Rate', ascending=False)

# Plot side-by-side bar plots
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

# Loss plot
rate_df_loss['Loss Rate'].plot(kind='bar', ax=axes[0], color='tomato')
axes[0].set_title('Genre-wise Loss Rate')
axes[0].set_ylabel('Probability')
axes[0].set_xlabel('Genre')
axes[0].tick_params(axis='x', rotation=90)
axes[0].grid(axis='y')

# Profit plot
rate_df_profit['Profit Rate'].plot(kind='bar', ax=axes[1], color='seagreen')
axes[1].set_title('Genre-wise Profit Rate')
axes[1].set_xlabel('Genre')
axes[1].tick_params(axis='x', rotation=90)
axes[1].grid(axis='y')

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



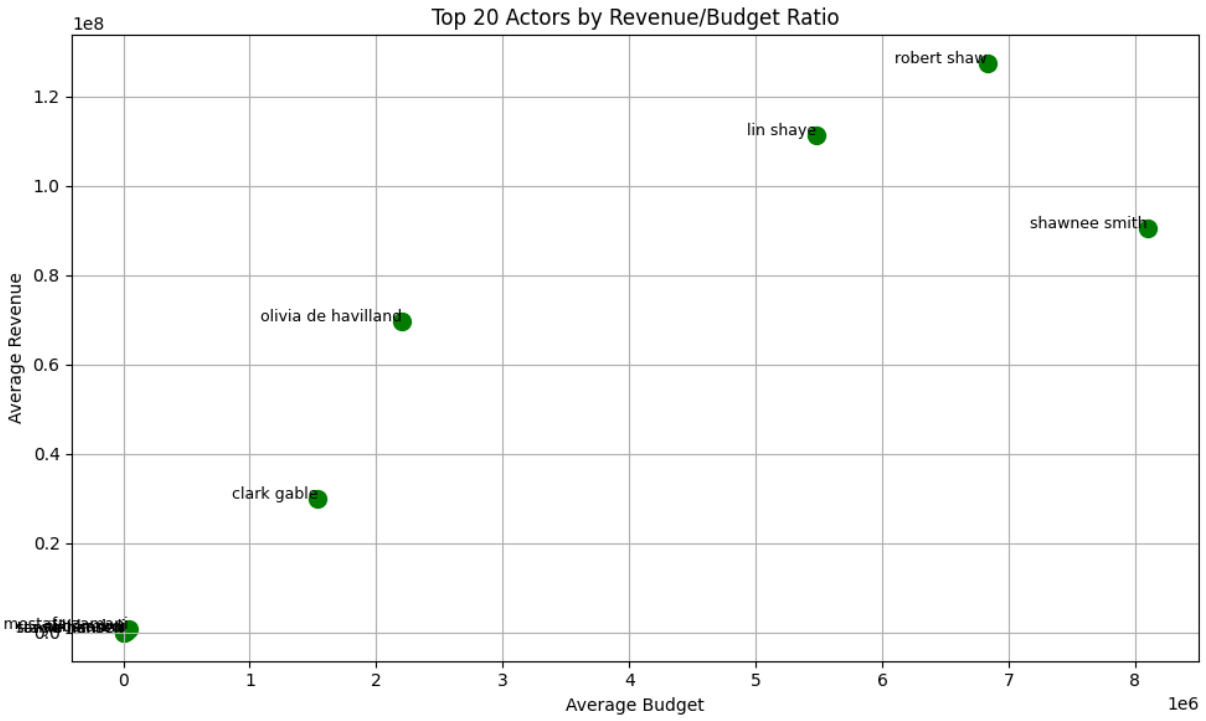
Visualization 7 - Top 20 actors on average revenue-to-budget ratio

From the plot, we can see Robert Shaw, Lin Shaye, and Shawnee Smith stand out with high average revenues, indicating that the movies they participated in typically generated substantial returns compared to their budgets. Olivia de Havilland and Clark Gable also show strong positive returns on relatively modest budgets. Several actors in the lower-left corner (e.g., Margaret Hamilton, Steve Landesberg) had low average budgets and revenues, suggesting lower overall commercial impact, but they still made the top 20 due to high revenue-to-budget efficiency in smaller productions.

In [200…

```
# Explode top_cast_list
df_exploded = df.explode('top_cast_list')
```

```
actor_avg = df_exploded.groupby('top_cast_list').agg(
    avg_budget=('budget', 'mean'),
    avg_revenue=('revenue', 'mean'),
    movie_count=('budget', 'count')
)

# Filter to only actors with 5+ movies to avoid outliers
actor_avg = actor_avg[actor_avg['movie_count'] >= 5]

actor_avg['ratio'] = actor_avg['avg_revenue'] / actor_avg['avg_budget']
top10 = actor_avg.sort_values('ratio', ascending=False).head(10)

plt.figure(figsize=(10, 6))
plt.scatter(top10['avg_budget'], top10['avg_revenue'], color='green', s=100)

for actor, row in top10.iterrows():
    plt.text(row['avg_budget'], row['avg_revenue'], actor, fontsize=9, ha='r

plt.xlabel('Average Budget')
plt.ylabel('Average Revenue')
plt.title('Top 20 Actors by Revenue/Budget Ratio')
plt.grid(True)
plt.tight_layout()
plt.show()
```



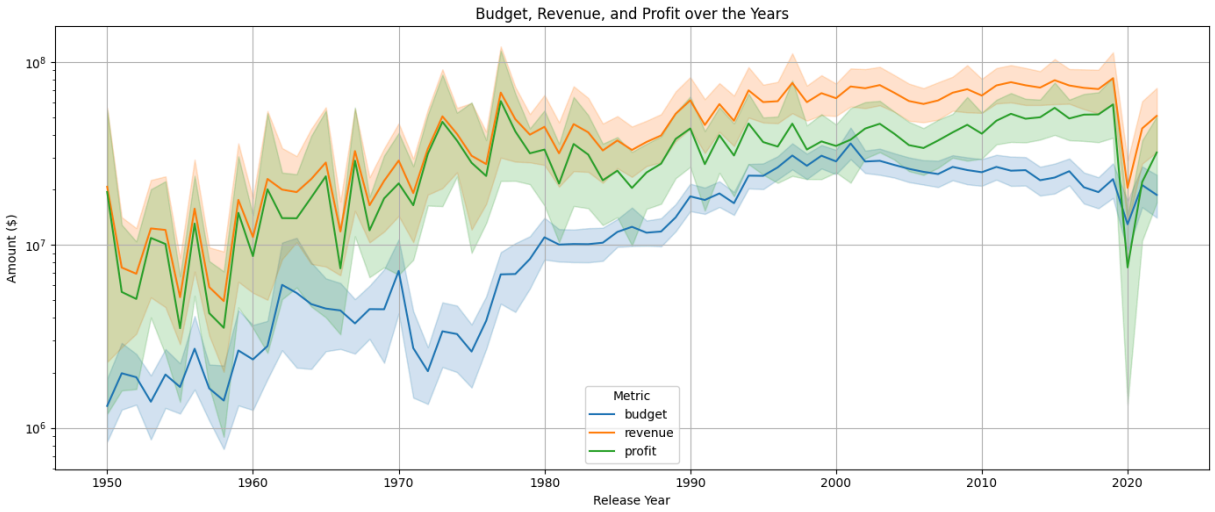
Visualization 8 - Trends of average budget, revenue, and profit

This line plot visualizes the trends of average budget, revenue, and profit for movies over the years from 1950 to 2022. Each line represents the yearly average for one financial metric—budget (blue), revenue (orange), and profit (green)—with shaded areas indicating variance (standard error). The Y-axis is on a logarithmic scale to accommodate the wide range of dollar amounts.

From the graph, we can observe A consistent upward trend in budget and revenue from the 1980s to the late 2010s, reflecting inflation and the rise of big-budget blockbusters. Profit (revenue minus budget) also follows a similar upward trend, peaking shortly before 2020. A sharp drop in all three metrics occurs around 2020, signaling the impact of the COVID-19 pandemic on film production and box office performance.

In [201…

```
df['release_year'] = df['release_date'].dt.year
df['profit'] = df['revenue'] - df['budget']
df_filtered = df[(df['release_year'] >= 1950) & (df['release_year'] <= 2022)

# Melt data into long format for seaborn
melted_df = df_filtered.melt(id_vars='release_year', value_vars=['budget', '
                             var_name='Metric', value_name='Amount')

# Plot
plt.figure(figsize=(14, 6))
sns.lineplot(data=melted_df, x='release_year', y='Amount', hue='Metric')
plt.title('Budget, Revenue, and Profit over the Years')
plt.xlabel('Release Year')
plt.ylabel('Amount ($)')
plt.yscale('log')  # optional for better visibility of scale differences
plt.grid(True)
plt.tight_layout()
plt.show()
```



Visualization 9 - correlation heatmap for the numerical features, specifically budget, revenue, vote_average, popularity, and vote_count.

Strong positive correlation (closer to 1), no correlation (0), strong negative correlation (closer to -1)

From the plot, we can see:

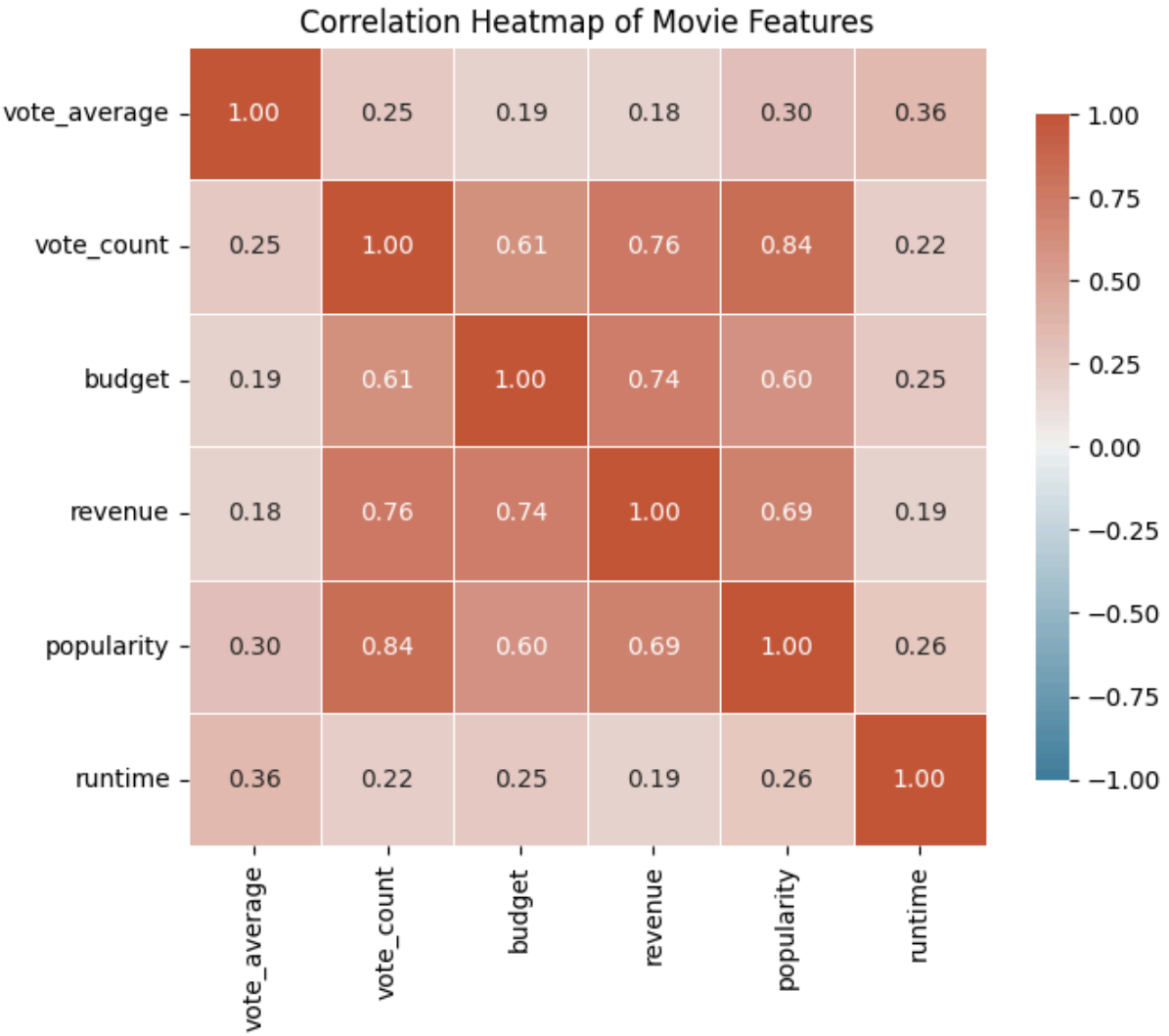
- Popularity and vote count are the strongest indicators of revenue.

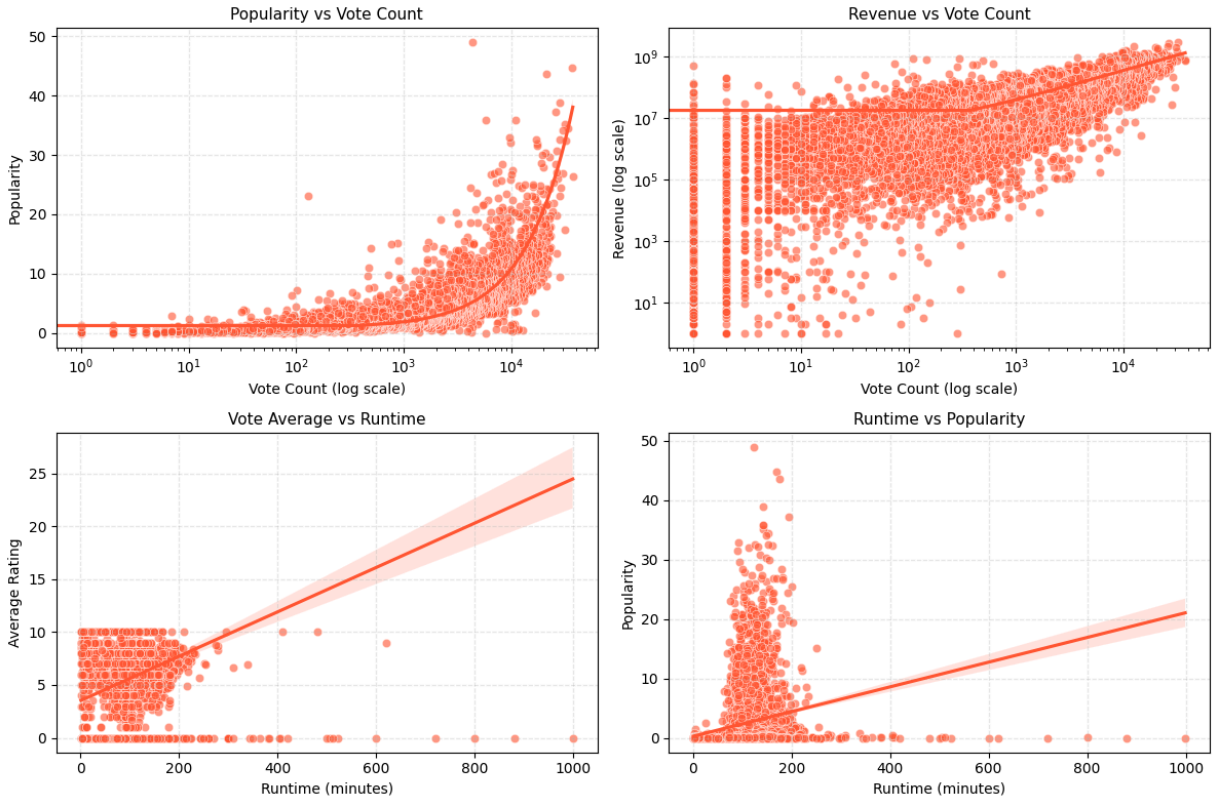
- Budget matters a lot — it correlates with both revenue and popularity.
- High vote average doesn't guarantee commercial success — critically acclaimed ≠ box office hit.
- Runtime is the least impactful among these features.The popularity score isn't strongly determined by runtime. However, very short movies (< 60 min) tend to have low popularity,

In [202…

```
# numerical columns for correlation
numerical_features = ['vote_average', 'vote_count', 'budget', 'revenue', 'po

# Correlation matrix Calculation
correlation_matrix = df[numerical_features].corr()

plt.figure(figsize=(7, 6))

# Custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix,
            annot=True,           # Show the correlation values
            fmt='.2f',            # Format to 2 decimal places
            cmap=cmap,            # Custom colormap
            vmax=1.0,             # Maximum correlation value
            vmin=-1.0,            # Minimum correlation value
            center=0,             # Center the colormap at zero
            square=True,          # Make the cells square
            linewidths=.5,        # Width of the lines between cells
            cbar_kws={"shrink": .8})  # Size of colorbar

plt.title('Correlation Heatmap of Movie Features')
plt.tight_layout()
plt.show()

# Sub-plots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))  # 2 rows, 2 columns

# 1. Popularity vs Vote Count (log X)
sns.scatterplot(data=df, x='vote_count', y='popularity', color='#FF5733', al
sns.regplot(data=df, x='vote_count', y='popularity', scatter=False, color='#
axes[0, 0].set_xscale('log')
axes[0, 0].set_title('Popularity vs Vote Count', fontsize=11)
axes[0, 0].set_xlabel('Vote Count (log scale)')
axes[0, 0].set_ylabel('Popularity')
axes[0, 0].grid(True, linestyle='--', alpha=0.3)

# 2. Revenue vs Vote Count (log-log)
sns.scatterplot(data=df, x='vote_count', y='revenue', color='#FF5733', alpha
sns.regplot(data=df, x='vote_count', y='revenue', scatter=False, color='#FF5
axes[0, 1].set_xscale('log')
axes[0, 1].set_yscale('log')
axes[0, 1].set_title('Revenue vs Vote Count', fontsize=11)
axes[0, 1].set_xlabel('Vote Count (log scale)')
axes[0, 1].set_ylabel('Revenue (log scale)')
```

```
axes[0, 1].grid(True, linestyle='--', alpha=0.3)

# 3. Vote Average vs Runtime (linear)
sns.scatterplot(data=df, x='runtime', y='vote_average', color='#FF5733', alp
sns.regplot(data=df, x='runtime', y='vote_average', scatter=False, color='#F
axes[1, 0].set_title('Vote Average vs Runtime', fontsize=11)
axes[1, 0].set_xlabel('Runtime (minutes)')
axes[1, 0].set_ylabel('Average Rating')
axes[1, 0].grid(True, linestyle='--', alpha=0.3)

# 4. Runtime vs Popularity (NEW)
sns.scatterplot(data=df, x='runtime', y='popularity', color='#FF5733', alpha
sns.regplot(data=df, x='runtime', y='popularity', scatter=False, color='#FF5
axes[1, 1].set_title('Runtime vs Popularity', fontsize=11)
axes[1, 1].set_xlabel('Runtime (minutes)')
axes[1, 1].set_ylabel('Popularity')
axes[1, 1].grid(True, linestyle='--', alpha=0.3)

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





Visualization 10 - bar chart showing movie revenues by release month

From the plot, we can see:

- June leads with $88.69B, clearly marked with a bold color and **SUMMER** annotation, summer is typically when blockbusters are released (e.g., Marvel movies, big-budget sequels), so this aligns with industry trends.
- January shows the lowest return: $23.89B, marked with hatching. This reflects the industry pattern — January is often a dump month where studios release lower-priority films post-holiday season.
- A red horizontal line shows the average monthly revenue: ~$55.91B
- Summer (May–July): Dominates the box office — all 3 months are above average, December (HOLIDAY): Also performs extremely well — second only to June.
- Winter (Jan–Feb): Low performance months, likely due to post-holiday drop and lack of major releases.
- Fall (Sep–Nov): Mixed — September is modest, while October and November perform better (possibly due to pre-Oscar releases and holiday hype).

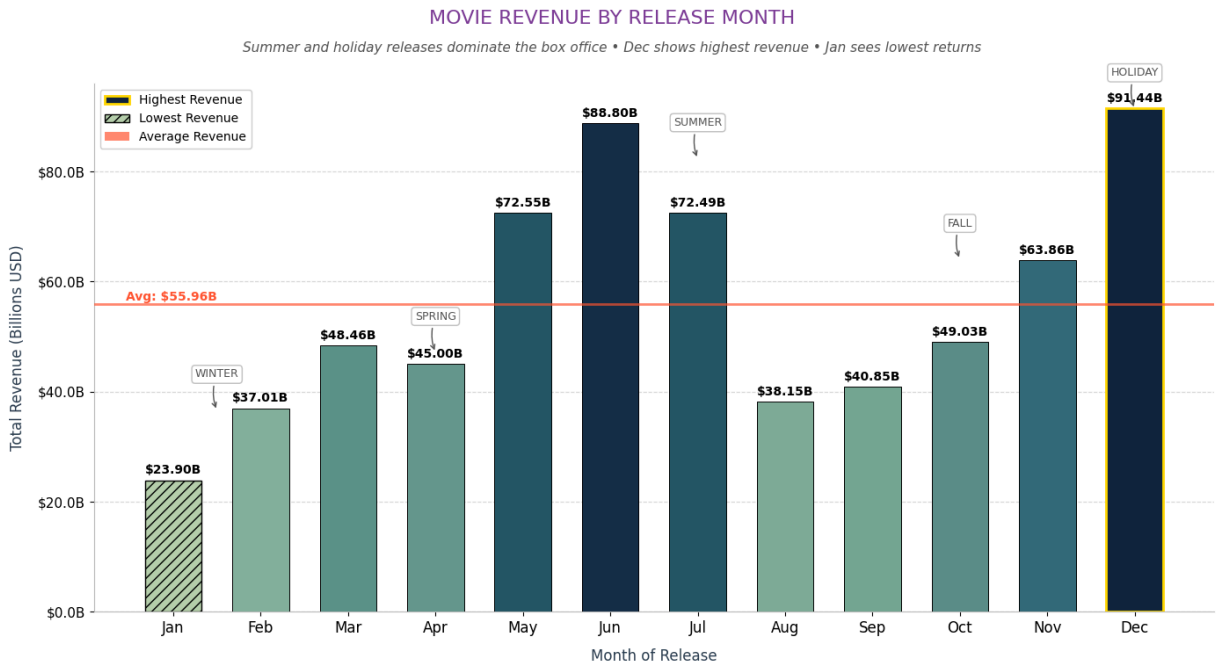
In [203…
```
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
df = df.dropna(subset=['release_date'])
```

```
df['month_num'] = df['release_date'].dt.month.astype(int)

# Grouping and summarizing revenue
month_revenue = df.groupby('month_num')['revenue'].sum().reset_index()
month_revenue['month_name'] = month_revenue['month_num'].apply(lambda x: cal
month_revenue.sort_values('month_num', inplace=True)

# ---------------------------
# 1)  Teal palette
# ---------------------------
teal_colors = [
    '#b5d1ae',  # Teal 1 (lightest)
    '#80ae9a',  # Teal 2
    '#568b87',  # Teal 3
    '#326b77',  # Teal 4
    '#1b485e',  # Teal 5
    '#122740'   # Teal 6 (darkest)
]

# A couple of helper functions to do gradient interpolation:
def hex_to_rgb(hex_color):
    """Convert hex string (e.g. '#568b87') to an (r,g,b) tuple scaled [0..1]
    hex_color = hex_color.lstrip('#')
    return tuple(int(hex_color[i:i+2], 16) / 255.0 for i in (0, 2, 4))

def rgb_to_hex(rgb_tuple):
    """Convert an (r,g,b) tuple scaled [0..1] back to a hex string."""
    return '#{:02x}{:02x}{:02x}'.format(
        int(round(rgb_tuple[0]*255)),
        int(round(rgb_tuple[1]*255)),
        int(round(rgb_tuple[2]*255))
    )

def interpolate_color(val, min_val, max_val, palette):
    """
    Interpolates a color from the given palette based on val's position
    between min_val and max_val. The palette is a list of hex colors
    going from lighter to darker teal.
    """
    if max_val == min_val:
        # Avoid division-by-zero if all values are the same
        return palette[-1]

    # Normalize fraction between 0 and 1
    fraction = (val - min_val) / (max_val - min_val)
    fraction *= (len(palette) - 1)

    idx_floor = int(np.floor(fraction))
    idx_ceil = min(idx_floor + 1, len(palette) - 1)
    local_frac = fraction - idx_floor

    c1 = hex_to_rgb(palette[idx_floor])
    c2 = hex_to_rgb(palette[idx_ceil])

    # Linear interpolate between two adjacent colors
```

```
    r = c1[0] + (c2[0] - c1[0]) * local_frac
    g = c1[1] + (c2[1] - c1[1]) * local_frac
    b = c1[2] + (c2[2] - c1[2]) * local_frac

    return rgb_to_hex((r, g, b))

# ------------------------------
# 2) Assign a color to each month
# ------------------------------
min_rev = month_revenue['revenue'].min()
max_rev = month_revenue['revenue'].max()

bar_colors = [
    interpolate_color(rev, min_rev, max_rev, teal_colors)
    for rev in month_revenue['revenue']
]

# ---------------------------
# 3) Plot with the new colors
# ---------------------------
fig, ax = plt.subplots(figsize=(14, 8), facecolor='white')
ax.set_facecolor('white')

x = np.arange(len(month_revenue['month_name']))
bar_width = 0.65

bars = ax.bar(
    x,
    month_revenue['revenue'] / 1e9,  # Convert to billions
    width=bar_width,
    color=bar_colors,
    edgecolor='black',
    linewidth=0.7,
    zorder=3
)

# Enhanced grid
ax.grid(axis='y', linestyle='--', alpha=0.3, color='gray', zorder=0)

# Highest and lowest revenue months
max_idx = month_revenue['revenue'].idxmax()
min_idx = month_revenue['revenue'].idxmin()

# Outline the highest bar in gold
bars[max_idx].set_edgecolor('gold')
bars[max_idx].set_linewidth(2)

# Hatch the lowest bar
bars[min_idx].set_hatch('///')
bars[min_idx].set_linewidth(1.0)

# Adding a horizontal average line
avg_revenue = month_revenue['revenue'].mean() / 1e9
ax.axhline(y=avg_revenue, color='#FF5733', linestyle='-', alpha=0.7, zorder=
ax.text(
    0.5, avg_revenue + 0.1,
```

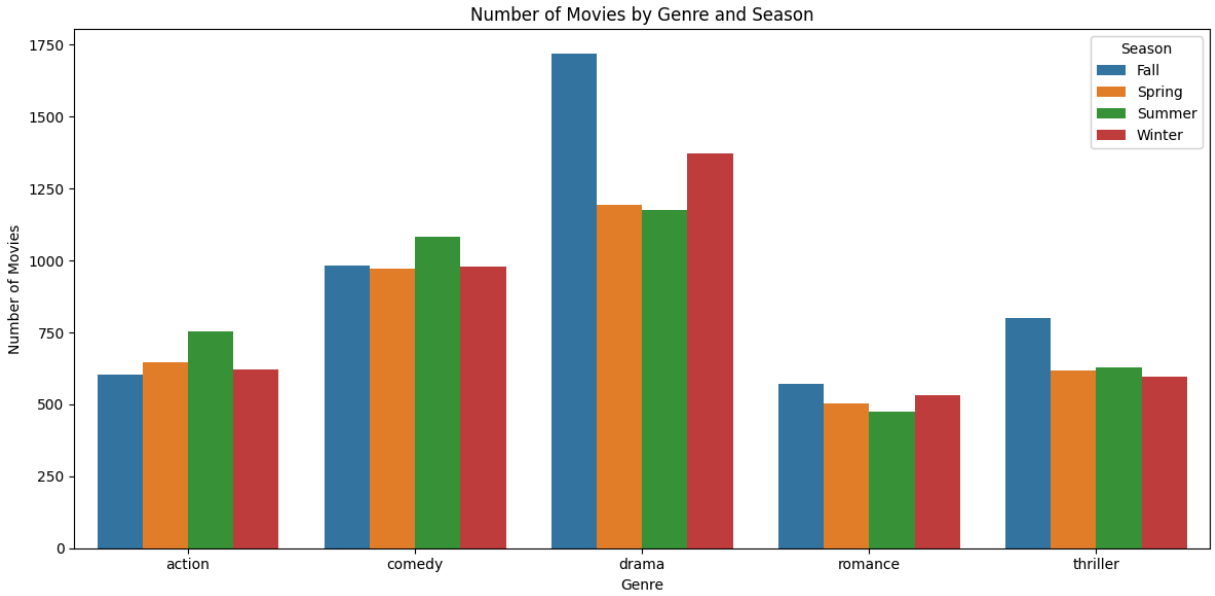
```
    f'Avg: ${avg_revenue:.2f}B',
    ha='right', va='bottom', color='#FF5733',
    fontweight='bold'
)

# Annotate each bar with revenue value
for i, bar in enumerate(bars):
    height = bar.get_height()
    ax.annotate(
        f'${height:.2f}B',
        xy=(bar.get_x() + bar.get_width()/2, height),
        xytext=(0, 3),
        textcoords='offset points',
        ha='center', va='bottom',
        fontsize=10, fontweight='bold',
        color='black'
    )

# Title and labels
plt.title("", fontsize=22, weight='bold', color='#2C3E50', loc='center', pac
plt.suptitle("MOVIE REVENUE BY RELEASE MONTH", fontsize=16, color='#7D3C98',

ax.set_xlabel("Month of Release", fontsize=12, labelpad=10, color='#2C3E50')
ax.set_ylabel("Total Revenue (Billions USD)", fontsize=12, labelpad=10, colo

ax.set_xticks(x)
ax.set_xticklabels(month_revenue['month_name'], fontsize=12)
ax.tick_params(axis='y', labelsize=11)

# Formatting y-axis with dollar signs
ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('${x:.1f}B'))

# Custom legend
legend_elements = [
    Patch(facecolor=bars[max_idx].get_facecolor(), edgecolor='gold', linewic
    Patch(facecolor=bars[min_idx].get_facecolor(), edgecolor='black', hatch=
    Patch(facecolor='#FF5733', alpha=0.7, label='Average Revenue')
]
ax.legend(handles=legend_elements, loc='best', frameon=True, framealpha=0.9)

# Adding a subtitle with analysis
highest_month = month_revenue.iloc[max_idx]['month_name']
lowest_month = month_revenue.iloc[min_idx]['month_name']
fig.text(
    0.5, 0.89,
    f"Summer and holiday releases dominate the box office • {highest_month}
    ha='center', fontsize=11, style='italic', color='#555555'
)

# Adding film strip decorative elements (optional — can remove if you like)
for i in range(8):
    x_pos = -0.8 + i * 0.4
    rect = plt.Rectangle((x_pos, -0.8), 0.25, 0.3, facecolor='black', alpha=
    ax.add_patch(rect)

    x_pos = 0.7 + i * 0.4
```

```
    rect = plt.Rectangle((x_pos, -0.8), 0.25, 0.3, facecolor='black', alpha=
    ax.add_patch(rect)

# Adding annotations for seasons (optional)
season_spans = [
    ((0, 1), "WINTER", 0.4),
    ((2, 4), "SPRING", 0.515),
    ((5, 7), "SUMMER", 0.9),
    ((8, 10), "FALL", 0.7),
    ((11, 11), "HOLIDAY", 0.999)
]

for (start, end), label, y_pos in season_spans:
    middle = (start + end) / 2
    ax.annotate(
        label,
        xy=(middle, y_pos * month_revenue['revenue'].max() / 1e9),
        xytext=(0, 25),
        textcoords='offset points',
        arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0.2', col
        ha='center', va='bottom',
        fontsize=9, color='#555555',
        bbox=dict(boxstyle="round,pad=0.3", facecolor='white', alpha=0.7, ec
    )

# Styling the spines
for spine in ['top', 'right']:
    ax.spines[spine].set_visible(False)
ax.spines['left'].set_color('#bbbbbb')
ax.spines['bottom'].set_color('#bbbbbb')

plt.tight_layout(rect=[0, 0, 1, 0.93])
plt.show()
```



In [204…

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

# Step 1: Ensure release_date and genres_list exist
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
df = df.dropna(subset=['release_date'])
df['genres_list'] = df['genres'].str.lower().str.strip().str.split(', ')
df_exploded = df.explode('genres_list')

# Step 2: Define seasons
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Fall'

df_exploded['season'] = df_exploded['release_date'].dt.month.apply(get_seasc

# Step 3: Filter to top genres (optional)
top_genres = df_exploded['genres_list'].value_counts().head(5).index
df_filtered = df_exploded[df_exploded['genres_list'].isin(top_genres)]

# Step 4: Group by genre and season
genre_season_counts = df_filtered.groupby(['genres_list', 'season']).size().

# Step 5: Plot using seaborn
plt.figure(figsize=(12, 6))
sns.barplot(data=genre_season_counts, x='genres_list', y='count', hue='seasc

plt.title('Number of Movies by Genre and Season')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.legend(title='Season')
plt.tight_layout()
plt.show()
```



Hypotheses

1. **Higher budgets correlate with higher revenues**
    - *Hypothesis:* Movies with larger budgets tend to generate more revenue.
    - *Tested by:* Log-log regression plot of Budget vs Revenue.
2. **Certain genres are more profitable on average**
    - *Hypothesis:* Genres like Adventure, Animation, and Science Fiction yield higher average revenue and profit than others.
    - *Tested by:* Bar plot of average revenue by genre, heatmap of profit per genre per year.
3. **Drama and Comedy dominate the movie industry by volume**
    - *Hypothesis:* Most movies produced belong to Drama and Comedy genres.
    - *Tested by:* Pie chart of movie counts by genre.
4. **The COVID-19 pandemic significantly impacted production and revenue**
    - *Hypothesis:* Movie release counts and total revenue declined sharply after 2019.
    - *Tested by:* Line plots of number of movies and total revenue per year, with COVID-19 marker.
5. **Actors with consistently high revenue-to-budget ratios indicate strong financial impact**
    - *Hypothesis:* Some actors consistently star in high-return movies.
    - *Tested by:* Top 20 actors by average revenue-to-budget ratio.
6. **Movie profitability varies significantly by genre**
    - *Hypothesis:* Certain genres like Family and Adventure have higher success rates, while genres like War and History have higher loss rates.

- *Tested by:* Profit/loss rate bar charts per genre.

7. **Movie releases during summer and holidays earn more**
    - *Hypothesis:* Movies released in May–July and December earn significantly more than others.
    - *Tested by:* Bar chart of total revenue by release month, with seasonal annotations.
8. **Vote count and popularity are the strongest correlates of revenue**
    - *Hypothesis:* A movie's popularity and vote count are strong indicators of revenue.
    - *Tested by:* Correlation heatmap and scatterplots.
9. **Genre trends evolve over time**
    - *Hypothesis:* The popularity of different genres shifts over the decades.
    - *Tested by:* Line plot showing number of movies per genre per year.
10. **Runtime has minimal effect on popularity and success**
    - *Hypothesis:* Movie length does not significantly influence revenue or popularity.
    - *Tested by:* Scatterplots of runtime vs vote average and popularity.

Conclusions

- **Budget and Revenue:** There is a strong positive correlation between a movie's budget and its revenue, though many low-budget movies underperform.
- **Genre Revenue:** Adventure, Animation, and Fantasy dominate in average revenue and profitability, while genres like War, Crime, and Horror are riskier investments.
- **Volume vs Profit:** Drama and Comedy are the most frequently produced genres but do not rank highest in profitability.
- **COVID-19 Impact:** There was a sharp decline in both movie releases and revenues starting in 2020.
- **Actor Efficiency:** Some actors consistently star in movies with high revenue-to-budget ratios, making them financially reliable choices.
- **Seasonality:** Summer (especially June) and December are the most lucrative months for movie releases, aligning with blockbuster and holiday seasons.
- **Correlation Insights:** Popularity and vote count are strong predictors of revenue, while vote average and runtime show weak or inconsistent influence.
- **Long-Term Trends:** The industry has seen steady growth in budget and revenue until 2020, with evolving genre preferences and production patterns over decades.

Reddit Movie Sentiment Analysis - Data Collection / Sentiment + Emotion Analysis Script

Author: Leonardo Ferreira

1. Objective

The main goal is to collect and analyze sentiment and emotional responses to movies using reddit comments and posts.

2. Sources

- **Reddit API**: Data source
- **Hugging Face Pretrained Models**:
  - Sentiment Analysis: Pretrained model for sentiment analysis
  - Emotion Recognition: Pretrained emotion recognition model

3. Reddit API setup

3.1 Create a reddit account

- If you don't already have one, go to reddit's registration page

3.2 Create a reddit application

- Go to your app preferences page while logged in.
- Scroll down to the bottom and click **"create another app"** (or **"create app"** if it's your first one).

3.3 Fill in the application details

- Select **"script"** as the application type.
- Provide a name for your application (e.g., "Movie Sentiment Analysis Project").
- Add a brief description.
- For the **"about url"** and **"redirect uri"** fields, you can use `http://localhost:8080` as a placeholder.
- Click **"create app"** to submit.

3.4 Get your credentials

- After creating the app, you'll see the **client ID** directly under the app name.
- The **client secret** will be displayed as **"secret"**.
- Make note of both, as you'll need them in your code.

3.5 Example: Initializing the reddit API with PRAW

```
import praw

reddit = praw.Reddit(
    client_id="YOUR_CLIENT_ID",
    client_secret="YOUR_CLIENT_SECRET",
    user_agent="python:movie.sentiment_emotion.analyzer:v1.0 (by
/u/your_username)"
)
```

4. Methodology

1. **Data Retrieval**
   - Utilize PRAW (python reddit API wrapper) to search reddit
   - Search parameters include:
     - Movie name as search query
     - Relevance sorting
     - Configurable time filter
     - Limit on number of posts
2. **Text Preprocessing**
   - Clean text by:
     - Converting to lowercase
     - Removing URLs
     - Eliminating non-alphabetic characters
     - Removing extra whitespace
   - Process both post titles and body text
   - Handle comments separately
3. **Sentiment Analysis**
   - Use CardiffNLP's RoBERTa-based sentiment model
   - Extract sentiment scores:
     - Negative sentiment
     - Neutral sentiment
     - Positive sentiment
     - Compound sentiment score
4. **Emotion Recognition**
   - Apply DistilRoBERTa emotion recognition model
   - Identify emotional categories:

        - Anger
        - Disgust
        - Fear
        - Joy
        - Neutral
        - Sadness
        - Surprise
5. **Data Storage**
    - Save processed data to CSV files
    - Separate files for posts and comments
    - Include:
        - Original text
        - Sentiment scores
        - Emotion scores
        - Metadata (author, timestamp, etc...)
6. **Basic EDA**
    - Sentiment comparison using a stacked bar chart
    - Emotion comparison using a grouped bar chart
    - Compound sentiment visualization through bar chart

5. ML models

Sentiment analysis model

- **Architecture**: RoBERTa
- **Training Data**: Twitter
- **Sentiment Categories**: 3 different sentiments (neutral, positive, negative)
- **Output**: Prob. distribution across sentiments
- **Compound Score**: Difference between positive and negative probs.

Emotion Recognition Model

- **Architecture**: DistilRoBERTa
- **Training Data**: English text
- **Emotion Categories**: 7 different emotions
- **Output**: Prob. distribution across emotions

6. Considerations

- **API Limitations**:
    - Implement rate limiting
- **Text Processing**:

- Truncate long texts to model's max length
- Clean and standardize text

In [ ]:
```
import praw
import pandas as pd
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
import time
import re
import os
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from scipy.special import softmax
# this is used to import private variables for reddit API
# you can either modify the script using your own credentials or create a .e
from dotenv import load_dotenv
```

In [ ]:
```
class RedditMovieDataCollector:
    def __init__(self, client_id, client_secret, user_agent):
        """
        initialize the reddit API client

        parameters:
        - client_id: your reddit API client ID
        - client_secret: your reddit API client secret
        - user_agent: unique identifier for your script
        """
        self.reddit = praw.Reddit(
            client_id=client_id,
            client_secret=client_secret,
            user_agent=user_agent
        )


        # initialize pretrained sentiment analysis model
        # using fine-tuned model for sentiment analysis
        # source: https://huggingface.co/cardiffnlp/twitter-roberta-base-ser
        self.sentiment_model_name = "cardiffnlp/twitter-roberta-base-sentime
        self.sentiment_tokenizer = AutoTokenizer.from_pretrained(self.sentim
        self.sentiment_model = AutoModelForSequenceClassification.from_pretr


        # doing the same but for emotion analysis
        # source: https://huggingface.co/j-hartmann/emotion-english-distilro
        self.emotion_model_name = "j-hartmann/emotion-english-distilroberta-
        self.emotion_tokenizer = AutoTokenizer.from_pretrained(self.emotion_
        self.emotion_model = AutoModelForSequenceClassification.from_pretrai

        # get emotion labels dynamically from the model configuration
        self.emotion_labels = [
            self.emotion_model.config.id2label[i]
            for i in range(len(self.emotion_model.config.id2label))
        ]
```

```
        # create emotion column names
        self.emotion_columns = [f"{label.lower()}_emotion" for label in self


        # create folder for data if it doesn't exist
        if not os.path.exists('movie_data_reddit'):
            os.makedirs('movie_data_reddit')

    def clean_text(self, text):
        """
        clean and preprocess text data

        parameters:
        - text: text to clean

        returns:
        - cleaned text
        """
        if text is None:
            return ""

        # to lowercase
        text = text.lower()

        # remove urls
        text = re.sub(r'http\S+', '', text)

        # keep only letters and spaces
        text = re.sub(r'[^a-zA-Z\s]', '', text)

        # remove extra whitespace
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def get_sentiment_scores(self, text):
        """
        calculate sentiment scores for a given text using VADER

        parameters:
        - text: text to analyze

        returns:
        - dictionary with sentiment scores
        """
        if not text:
            # return neutral sentiment if text is empty
            return {
                'compound': 0,
                'pos': 0,
                'neu': 1,
                'neg': 0
            }

        # truncate text if it's too long for the model
        max_length = 512
```

```
        if len(text) > max_length:
            text = text[:max_length]

        encoded_input = self.sentiment_tokenizer(text, return_tensors='pt',

        # get model output
        with torch.no_grad():
            output = self.sentiment_model(**encoded_input)

        # get probabilities
        scores = softmax(output.logits[0].numpy())

        # map scores to sentiment categories (negative, neutral, positive)
        sentiment_scores = {
            'neg': float(scores[0]),
            'neu': float(scores[1]),
            'pos': float(scores[2]),
            'compound': float(scores[2] - scores[0])
        }

        return sentiment_scores

    def get_emotion_scores(self, text):
        """
        calculate emotion scores

        parameters:
        - text: text to analyze

        returns:
        - dictionary with emotion scores
        """
        if not text:
            # return neutral emotion scores if text is empty
            return {label.lower(): 0 for label in self.emotion_labels}

        # truncate text if it's too long for the model
        max_length = 512
        if len(text) > max_length:
            text = text[:max_length]

        encoded_input = self.emotion_tokenizer(text, return_tensors='pt', tr

        # get model output
        with torch.no_grad():
            output = self.emotion_model(**encoded_input)

        # get probabilities
        scores = softmax(output.logits[0].numpy())

        # map scores to emotion categories dynamically
        emotion_mapping = {
            self.emotion_model.config.id2label[i].lower(): float(scores[i])
            for i in range(len(self.emotion_labels))
        }
```

```
        return emotion_mapping

    def get_date_range(self, release_date, months_before):
        """
        calculate a date range starting N months before a movie's release da

        parameters:
        - release_date: release date datetime
        - months_before: N of months before release date

        returns:
        - start_date: datetime for the start date
        - end_date: datetime for the end date (release date)
        """
        # calculate start date (N months before release)
        start_date = release_date - timedelta(days=30 * months_before)

        return start_date, release_date

    def collect_reddit_data(self, movie_name, release_date, months_before=3,
        """
        collect reddit data for a specific movie

        parameters:
        - movie_name: name of the movie to search for
        - release_date: movie's release date
        - months_before: number of months before release date
        - limit: max number of posts

        returns:
        - df with collected data
        """

        # calculate date range
        start_date, end_date = self.get_date_range(release_date, months_befc

        # convert dates to unix timestamps
        start_timestamp = int(start_date.timestamp())
        end_timestamp = int(end_date.timestamp())

        posts_data = []
        comments_data = []

        # search for posts related to the movie
        search_query = movie_name

        # to avoid infinite search we have to set a reasonable max limit to
        max_posts_to_check = limit * 100
        posts_processed = 0
        post_count = 0
        comment_count = 0

        for post in self.reddit.subreddit("all").search(search_query, sort="

            # we need to break if checked too many posts without finding enc
            if posts_processed > max_posts_to_check:
```

```
                print(f"maximum posts to check ({max_posts_to_check}) reache
                break

            # we skip posts outside the date range we stablished
            post_timestamp = post.created_utc
            if post_timestamp < start_timestamp or post_timestamp > end_time
                continue

            # if we reach the limit, then we can stop
            if post_count >= limit:
                break

            post_count += 1

            # clean title and text
            # for posts we use the TITLE and the POST TEXTUAL CONTENT for se
            clean_title = self.clean_text(post.title)
            clean_text = self.clean_text(post.selftext)
            combined_text = f"{clean_title} {clean_text}"

            # get sentiment scores
            sentiment_scores = self.get_sentiment_scores(combined_text)

            # get emotion scores
            emotion_scores = self.get_emotion_scores(combined_text)

            # process post
            post_data = {
                'id': post.id,
                'title': post.title,
                'text': post.selftext,
                'author': str(post.author),
                'score': post.score,
                'created_utc': datetime.fromtimestamp(post.created_utc),
                'subreddit': post.subreddit.display_name,
                'num_comments': post.num_comments,
                'compound_sentiment': sentiment_scores['compound'],
                'positive_sentiment': sentiment_scores['pos'],
                'neutral_sentiment': sentiment_scores['neu'],
                'negative_sentiment': sentiment_scores['neg'],
                **{f"{k}_emotion": v for k, v in emotion_scores.items()},
                'content_type': 'post'
            }
            posts_data.append(post_data)

            # get comments while skipping loading more comments to avoid API
            post.comments.replace_more(limit=0)
            for comment in post.comments.list():
                # filter comments by date as well
                comment_timestamp = comment.created_utc
                if comment_timestamp < start_timestamp or comment_timestamp
                    continue
                comment_count += 1

                # clean comment text
                # for commentws we use the COMMENT TEXTUAL BODY for sentimen
```

```
                clean_comment = self.clean_text(comment.body)

                # get sentiment scores
                comment_sentiment = self.get_sentiment_scores(clean_comment)

                # get emotion scores
                comment_emotion = self.get_emotion_scores(clean_comment)

                # process comment
                comment_data = {
                    'id': comment.id,
                    'post_id': post.id,
                    'text': comment.body,
                    'author': str(comment.author),
                    'score': comment.score,
                    'created_utc': datetime.fromtimestamp(comment.created_ut
                    'subreddit': post.subreddit.display_name,
                    'compound_sentiment': comment_sentiment['compound'],
                    'positive_sentiment': comment_sentiment['pos'],
                    'neutral_sentiment': comment_sentiment['neu'],
                    'negative_sentiment': comment_sentiment['neg'],
                    **{f"{k}_emotion": v for k, v in comment_emotion.items()
                    'content_type': 'comment'
                }
                comments_data.append(comment_data)

            # sleep to avoid rate limits
            time.sleep(0.5)

        # create dfs
        posts_df = pd.DataFrame(posts_data)
        comments_df = pd.DataFrame(comments_data)

        # save to csv
        movie_name_cleaned = movie_name.replace(" ", "_").lower()

        posts_filename = f"movie_data_reddit/{movie_name_cleaned}_posts.csv"
        comments_filename = f"movie_data_reddit/{movie_name_cleaned}_comment

        posts_df.to_csv(posts_filename, index=False, quoting=1, escapechar='
        comments_df.to_csv(comments_filename, index=False, quoting=1, escape

        print(f"Data collection is complete: {post_count} posts and {comment
        print(f"Data saved to {posts_filename} and {comments_filename}")

        # combine data for analysis
        all_data = pd.concat([posts_df, comments_df])

        return all_data

    def analyze_sentiment_and_emotion_distribution(self, data):
        """
        analyze the sentiment and emotion distribution of collected data

        parameters:
        - data: df with collected data
```

```
    returns:
    - df with sentiment and emotion distribution analysis
    """
    # calculate averages for sentiment scores
    sentiment_avg = {
        'Average compound score': data['compound_sentiment'].mean(),
        'Average positive score': data['positive_sentiment'].mean(),
        'Average neutral score': data['neutral_sentiment'].mean(),
        'Average negative score': data['negative_sentiment'].mean()
    }

    # categorize sentiments
    data['dominant_sentiment'] = data[['negative_sentiment', 'neutral_se

    # map the column names to more readable sentiment names
    sentiment_name_mapping = {
        'negative_sentiment': 'Negative',
        'neutral_sentiment': 'Neutral',
        'positive_sentiment': 'Positive'
    }

    # apply the mapping
    data['sentiment_category'] = data['dominant_sentiment'].map(sentimer

    # count by category
    sentiment_counts = data['sentiment_category'].value_counts().to_dict

    # calculate percentages
    total = sum(sentiment_counts.values())
    sentiment_percentages = {k: (v / total) * 100 for k, v in sentiment_

    # emotion analysis
    emotion_columns = [col for col in data.columns if col.endswith('_emo

    # calculate average emotion scores
    emotion_avg = {f'Average {col.split("_")[0]} emotion': data[col].mea

    # calculate total emotion scores across all content
    total_emotion_scores = {}
    for col in emotion_columns:
        emotion_name = col.split('_')[0]
        total_emotion_scores[emotion_name] = data[col].sum()

    # sort emotions
    sorted_emotions = sorted(total_emotion_scores.items(), key=lambda x:

    emotions_dict = {
        'emotions': [emotion for emotion, score in sorted_emotions],
        'emotions_scores': {emotion: score for emotion, score in sorted_
    }

    # calculate percentages for top 5 emotions
    total_emotion_score = sum(total_emotion_scores.values())
    emotions_percentages = {
        emotion: (score / total_emotion_score) * 100
```

```
            for emotion, score in emotions_dict['emotions_scores'].items()
        }
        emotions_dict['emotions_percentages'] = emotions_percentages

        # combine results
        analysis_result = {
            'sentiment_avg': sentiment_avg,
            'sentiment_counts': sentiment_counts,
            'sentiment_percentages': sentiment_percentages,
            'emotion_avg': emotion_avg,
            'emotions': emotions_dict
        }

        return analysis_result
```

In [3]:
```
# load credentials from .env file
load_dotenv()
REDDIT_CLIENT_ID = os.getenv("REDDIT_CLIENT_ID")
REDDIT_CLIENT_SECRET = os.getenv("REDDIT_CLIENT_SECRET")
REDDIT_USER_AGENT = os.getenv("REDDIT_USER_AGENT")

# initialize collector with reddit credentials
collector = RedditMovieDataCollector(REDDIT_CLIENT_ID, REDDIT_CLIENT_SECRET,
```

```
Some weights of the model checkpoint at cardiffnlp/twitter-roberta-base-sent
iment-latest were not used when initializing RobertaForSequenceClassificatio
n: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
- This IS expected if you are initializing RobertaForSequenceClassification
from the checkpoint of a model trained on another task or with another archi
tecture (e.g. initializing a BertForSequenceClassification model from a Bert
ForPreTraining model).
- This IS NOT expected if you are initializing RobertaForSequenceClassificat
ion from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a BertForSequenceCl
assification model).
```

In [4]:
```
# collecting data for Oppenheimer
# collect data for a given movie
# change the movie name to get sentiments about different movies
movie_name = "Oppenheimer"
oppenheimer_release = datetime(2023, 7, 21)
# limit = 50 means: 50 posts. Be aware that a post can have multiple comment
# moreover, since the API does not provide a way to search between date rang
# the algorithm will try up to limit * 100 searches to reach the number of p
# however, if it does not reach it will break to avoid an infinite search
limit = 50
months_before = 3
oppenheimer_data = collector.collect_reddit_data(movie_name, oppenheimer_rel
```

```
Data collection is complete: 50 posts and 16997 comments
Data saved to movie_data_reddit/oppenheimer_posts.csv and movie_data_reddit/
oppenheimer_comments.csv
```

In [5]:
```
"""
Metrics meaning
```

```
Sentiment scores:
Average compound score: overall sentiment score, ranging from -1 (very negat
Average positive score: represents the proportion of text that expresses pos
Average neutral score: % of the content that is presenting emotionally neutr
Average negative score: % of the content that is presenting emotionally nega

Sentiment distribution:
Neutral: number of items expressing sentiments classified as neither positiv
Positive: number of items expressing positive sentiments
Very positive: number of items expressing strongly positive sentiments
Negative: number of items expressing negative sentiments
Very negative: number of items expressing strongly negative sentiments
"""
# brief analysis of sentiment and emotion distribution for the given movie
oppenheimer_analysis = collector.analyze_sentiment_and_emotion_distribution(

print("Sentiment analysis results:")
print("Average sentiment scores:")
for k, v in oppenheimer_analysis['sentiment_avg'].items():
    print(f"  {k}: {v:.4f}")

print("\n")

print("Sentiment distribution:")
for k, v in oppenheimer_analysis['sentiment_counts'].items():
    percentage = oppenheimer_analysis['sentiment_percentages'][k]
    print(f"  {k}: {v} ({percentage:.2f}%)")
```

```
Sentiment analysis results:
Average sentiment scores:
  Average compound score: -0.1287
  Average positive score: 0.2230
  Average neutral score: 0.4253
  Average negative score: 0.3517


Sentiment distribution:
  Neutral: 7699 (45.16%)
  Negative: 6134 (35.98%)
  Positive: 3214 (18.85%)
```

In [6]:

```
# available emotions
print("\nAvailable emotions in the pretrained model:")
for i, label in enumerate(collector.emotion_labels):
    print(f"{i}: {label}")

# emotion analysis results
print("Emotion analysis results:")
print("Average emotion scores:")
for k, v in oppenheimer_analysis['emotion_avg'].items():
    print(f"  {k}: {v:.4f}")

print("\n")

print("Top emotions:")
emotions_data = oppenheimer_analysis['emotions']
```

```
for emotion, score in zip(
    emotions_data['emotions'],
    emotions_data['emotions_scores'].values()
):
    percentage = emotions_data['emotions_percentages'][emotion]
    print(f"  {emotion.capitalize()}: {score:.2f} ({percentage:.2f}%)")
```

```
Available emotions in the pretrained model:
0: anger
1: disgust
2: fear
3: joy
4: neutral
5: sadness
6: surprise
Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.1094
  Average disgust emotion: 0.0402
  Average fear emotion: 0.0537
  Average joy emotion: 0.1560
  Average neutral emotion: 0.3085
  Average sadness emotion: 0.1206
  Average surprise emotion: 0.2064


Top emotions:
  Neutral: 5258.96 (31.01%)
  Surprise: 3518.86 (20.75%)
  Joy: 2660.05 (15.69%)
  Sadness: 2056.03 (12.12%)
  Anger: 1865.00 (11.00%)
  Fear: 915.62 (5.40%)
  Disgust: 684.49 (4.04%)
```

In [7]:

```
# collecting data for Deadpool & Wolverine
movie_name = "Deadpool & Wolverine"
deadpool_wolverine_release = datetime(2024, 7, 26)
deadwolv_data = collector.collect_reddit_data(movie_name, deadpool_wolverine

deadwolv_analysis = collector.analyze_sentiment_and_emotion_distribution(dea

print("Sentiment analysis results:")
print("Average sentiment scores:")
for k, v in deadwolv_analysis['sentiment_avg'].items():
    print(f"  {k}: {v:.4f}")

print("\n")

print("Sentiment distribution:")
for k, v in deadwolv_analysis['sentiment_counts'].items():
    percentage = deadwolv_analysis['sentiment_percentages'][k]
    print(f"  {k}: {v} ({percentage:.2f}%)")

print("\nAvailable emotions in the pretrained model:")
for i, label in enumerate(collector.emotion_labels):
```

```
    print(f"{i}: {label}")

print("Emotion analysis results:")
print("Average emotion scores:")
for k, v in deadwolv_analysis['emotion_avg'].items():
    print(f"  {k}: {v:.4f}")

print("\n")

print("Top emotions:")
emotions_data = deadwolv_analysis['emotions']
for emotion, score in zip(
    emotions_data['emotions'],
    emotions_data['emotions_scores'].values()
):
    percentage = emotions_data['emotions_percentages'][emotion]
    print(f"  {emotion.capitalize()}: {score:.2f} ({percentage:.2f}%)")
```

```
Data collection is complete: 50 posts and 14877 comments
Data saved to movie_data_reddit/deadpool_&_wolverine_posts.csv and movie_dat
a_reddit/deadpool_&_wolverine_comments.csv
Sentiment analysis results:
Average sentiment scores:
  Average compound score: -0.0576
  Average positive score: 0.2499
  Average neutral score: 0.4425
  Average negative score: 0.3075


Sentiment distribution:
  Neutral: 7109 (47.63%)
  Negative: 4543 (30.43%)
  Positive: 3275 (21.94%)

Available emotions in the pretrained model:
0: anger
1: disgust
2: fear
3: joy
4: neutral
5: sadness
6: surprise
Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.0834
  Average disgust emotion: 0.0358
  Average fear emotion: 0.0349
  Average joy emotion: 0.1870
  Average neutral emotion: 0.3100
  Average sadness emotion: 0.1255
  Average surprise emotion: 0.2190


Top emotions:
  Neutral: 4627.44 (31.14%)
  Surprise: 3268.94 (22.00%)
  Joy: 2791.35 (18.78%)
  Sadness: 1873.88 (12.61%)
  Anger: 1244.39 (8.37%)
  Disgust: 533.99 (3.59%)
  Fear: 521.01 (3.51%)
```

In [8]:
```
# collecting data for Flash (2023)
movie_name = "Flash (2023)"
flash_release = datetime(2023, 6, 16)
flash_data = collector.collect_reddit_data(movie_name, flash_release, months

flash_analysis = collector.analyze_sentiment_and_emotion_distribution(flash_

print("Sentiment analysis results:")
print("Average sentiment scores:")
for k, v in flash_analysis['sentiment_avg'].items():
    print(f"  {k}: {v:.4f}")
```

```
print("\n")

print("Sentiment distribution:")
for k, v in flash_analysis['sentiment_counts'].items():
    percentage = flash_analysis['sentiment_percentages'][k]
    print(f"  {k}: {v} ({percentage:.2f}%)")

print("\nAvailable emotions in the pretrained model:")
for i, label in enumerate(collector.emotion_labels):
    print(f"{i}: {label}")

print("Emotion analysis results:")
print("Average emotion scores:")
for k, v in flash_analysis['emotion_avg'].items():
    print(f"  {k}: {v:.4f}")

print("\n")

print("Top emotions:")
emotions_data = flash_analysis['emotions']
for emotion, score in zip(
    emotions_data['emotions'],
    emotions_data['emotions_scores'].values()
):
    percentage = emotions_data['emotions_percentages'][emotion]
    print(f"  {emotion.capitalize()}: {score:.2f} ({percentage:.2f}%)")
```

```
Data collection is complete: 31 posts and 7252 comments
Data saved to movie_data_reddit/flash_(2023)_posts.csv and movie_data_reddi
t/flash_(2023)_comments.csv
Sentiment analysis results:
Average sentiment scores:
  Average compound score: -0.2093
  Average positive score: 0.1895
  Average neutral score: 0.4118
  Average negative score: 0.3987


Sentiment distribution:
  Negative: 3084 (42.35%)
  Neutral: 3028 (41.58%)
  Positive: 1171 (16.08%)

Available emotions in the pretrained model:
0: anger
1: disgust
2: fear
3: joy
4: neutral
5: sadness
6: surprise
Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.1200
  Average disgust emotion: 0.0493
  Average fear emotion: 0.0577
  Average joy emotion: 0.1368
  Average neutral emotion: 0.3150
  Average sadness emotion: 0.1305
  Average surprise emotion: 0.1861


Top emotions:
  Neutral: 2293.99 (31.65%)
  Surprise: 1355.52 (18.70%)
  Joy: 996.21 (13.74%)
  Sadness: 950.11 (13.11%)
  Anger: 873.82 (12.05%)
  Fear: 420.32 (5.80%)
  Disgust: 359.03 (4.95%)
```

In [14]:
```
# collecting data for Joker: Folie à Deux
movie_name = "Joker: Folie a Deux"
joker_release = datetime(2024, 10, 4)
joker_data = collector.collect_reddit_data(movie_name, joker_release, months

joker_analysis = collector.analyze_sentiment_and_emotion_distribution(joker_

print("Sentiment analysis results:")
print("Average sentiment scores:")
for k, v in joker_analysis['sentiment_avg'].items():
    print(f"  {k}: {v:.4f}")
```

```
print("\n")

print("Sentiment distribution:")
for k, v in joker_analysis['sentiment_counts'].items():
    percentage = joker_analysis['sentiment_percentages'][k]
    print(f"  {k}: {v} ({percentage:.2f}%)")

print("\nAvailable emotions in the pretrained model:")
for i, label in enumerate(collector.emotion_labels):
    print(f"{i}: {label}")

print("Emotion analysis results:")
print("Average emotion scores:")
for k, v in joker_analysis['emotion_avg'].items():
    print(f"  {k}: {v:.4f}")

print("\n")

print("Top emotions:")
emotions_data = joker_analysis['emotions']
for emotion, score in zip(
    emotions_data['emotions'],
    emotions_data['emotions_scores'].values()
):
    percentage = emotions_data['emotions_percentages'][emotion]
    print(f"  {emotion.capitalize()}: {score:.2f} ({percentage:.2f}%)")
```

```
Data collection is complete: 50 posts and 9070 comments
Data saved to movie_data_reddit/joker:_folie_a_deux_posts.csv and movie_data
_reddit/joker:_folie_a_deux_comments.csv
Sentiment analysis results:
Average sentiment scores:
  Average compound score: -0.2138
  Average positive score: 0.1956
  Average neutral score: 0.3951
  Average negative score: 0.4094


Sentiment distribution:
  Negative: 3992 (43.77%)
  Neutral: 3658 (40.11%)
  Positive: 1470 (16.12%)

Available emotions in the pretrained model:
0: anger
1: disgust
2: fear
3: joy
4: neutral
5: sadness
6: surprise
Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.1026
  Average disgust emotion: 0.0458
  Average fear emotion: 0.0475
  Average joy emotion: 0.1802
  Average neutral emotion: 0.2690
  Average sadness emotion: 0.1426
  Average surprise emotion: 0.2027


Top emotions:
  Neutral: 2452.86 (27.16%)
  Surprise: 1848.66 (20.47%)
  Joy: 1643.45 (18.20%)
  Sadness: 1300.94 (14.40%)
  Anger: 935.57 (10.36%)
  Fear: 432.85 (4.79%)
  Disgust: 417.67 (4.62%)
```

In [50]:
```python
# combine all movie datasets
def combine_movie_data():
    movies_data = {
        'Oppenheimer': oppenheimer_data,
        'Deadpool & Wolverine': deadwolv_data,
        'Flash (2023)': flash_data,
        'Joker: Folie a Deux': joker_data
    }
    return movies_data
```

In [51]:
```python
# sentiment comparison bar chart
def plot_sentiment_comparison(movies_data):
```

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

    sentiment_data = []
    for movie, data in movies_data.items():
        sentiment_dist = data[['negative_sentiment', 'neutral_sentiment', 'p
        sentiment_data.append({
            'Movie': movie,
            'Negative': sentiment_dist['negative_sentiment'],
            'Neutral': sentiment_dist['neutral_sentiment'],
            'Positive': sentiment_dist['positive_sentiment']
        })

    sentiment_df = pd.DataFrame(sentiment_data)

    sentiment_df.set_index('Movie')[['Negative', 'Neutral', 'Positive']].plo
    plt.title('Sentiment distribution across movies', fontsize=15)
    plt.xlabel('Movies', fontsize=12)
    plt.ylabel('Average sentiment scores', fontsize=12)
    plt.legend(title='Sentiment', bbox_to_anchor=(1.05, 1), loc='upper left'
    plt.tight_layout()
    plt.show()
```

In [58]:
```
# emotion comparison grouped bar chart
def plot_emotion_comparison(movies_data):
    plt.figure(figsize=(14, 7))

    emotion_columns = ['anger_emotion', 'disgust_emotion', 'fear_emotion',
                       'joy_emotion', 'neutral_emotion', 'sadness_emotion',

    emotion_data = []
    for movie, data in movies_data.items():
        emotion_means = data[emotion_columns].mean()
        emotion_entry = {'Movie': movie, **emotion_means}
        emotion_data.append(emotion_entry)

    emotion_df = pd.DataFrame(emotion_data)

    emotion_df_melted = emotion_df.melt(id_vars=['Movie'],
                                        var_name='Emotion',
                                        value_name='Average Score')

    plt.figure(figsize=(15, 8))
    sns.barplot(x='Emotion', y='Average Score', hue='Movie', data=emotion_df
    plt.title('Emotion scores across movies', fontsize=15)
    plt.xlabel('Emotions', fontsize=12)
    plt.ylabel('Average emotion scores', fontsize=12)
    plt.xticks(rotation=45)
    plt.legend(title='Movie', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()
```

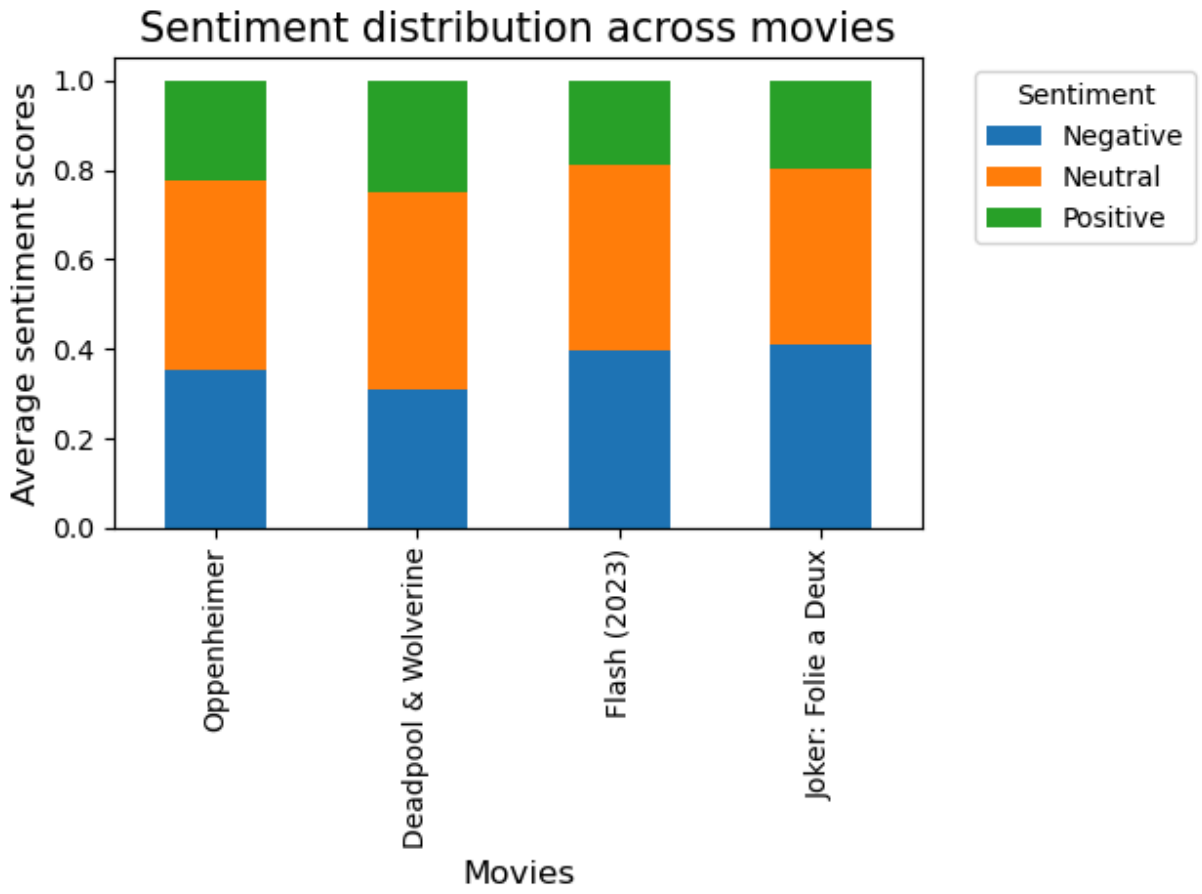
In [53]:
```
# compound sentiment analysis
def analyze_compound_sentiment(movies_data):
    plt.figure(figsize=(12, 7))
```

```
    compound_sentiments = {}
    for movie, data in movies_data.items():
        compound_mean = data['compound_sentiment'].mean()
        compound_sentiments[movie] = compound_mean

    # create bar plot with green bars being positive values and red negative
    colors = ['green' if val >= 0 else 'red' for val in compound_sentiments.
    bars = plt.bar(compound_sentiments.keys(), compound_sentiments.values(),
    plt.title('Average compound sentiment across movies', fontsize=15)
    plt.xlabel('Movies', fontsize=12)
    plt.ylabel('Average compound sentiment', fontsize=12)
    plt.axhline(y=0, color='black', linestyle='--')

    # add value labels on top of each bar since values can be negative
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height,
                 f'{height:.4f}',
                 ha='center', va='bottom' if height >= 0 else 'top')

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

In [54]:
```
# execute data combination
movies_data = combine_movie_data()
```

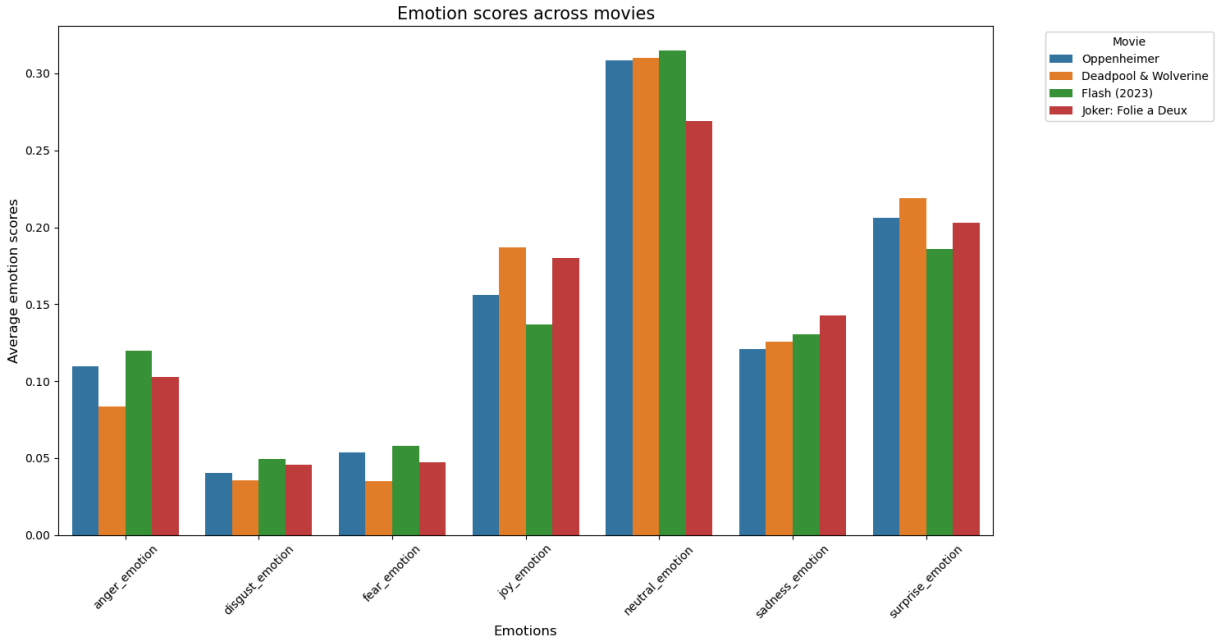
In [55]:
```
plot_sentiment_comparison(movies_data)
```

```
<Figure size 1200x600 with 0 Axes>
```

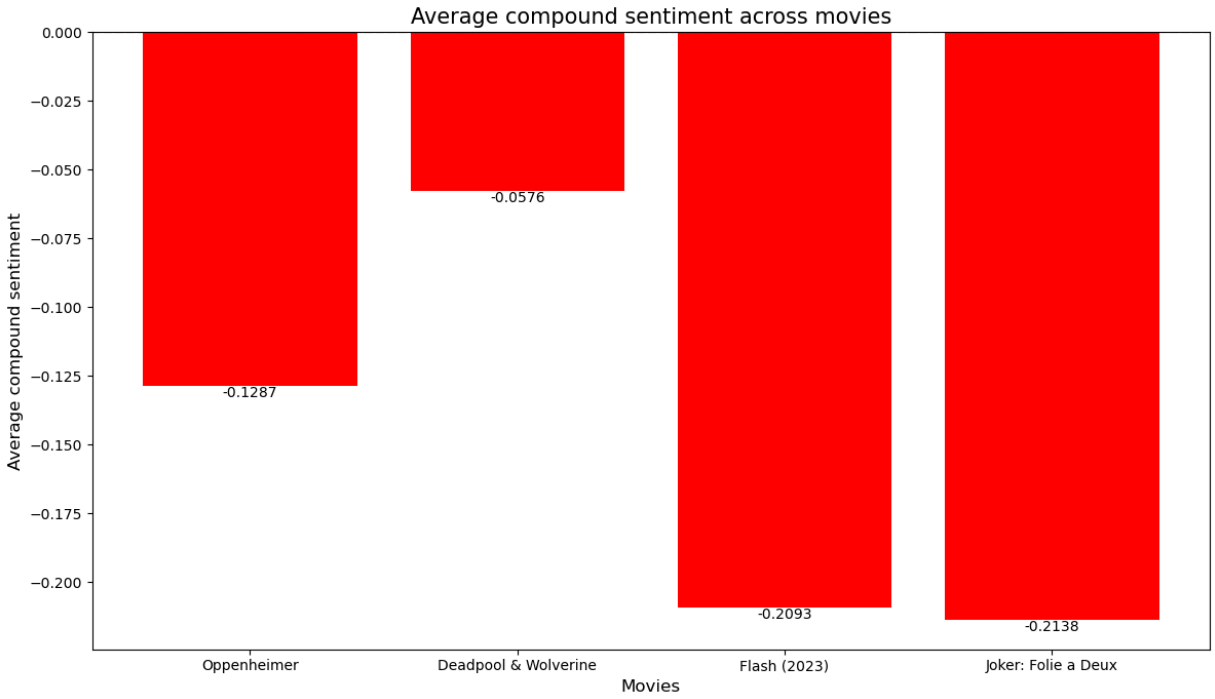


In [59]:
```
plot_emotion_comparison(movies_data)
```

```
<Figure size 1400x700 with 0 Axes>
```



In [57]:
```
compound_sentiments = analyze_compound_sentiment(movies_data)
```


Average compound sentiment across movies
Average compound sentiment
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
−0.175
−0.200
-0.1287
-0.0576
-0.2093
-0.2138
Oppenheimer
Deadpool & Wolverine
Flash (2023)
Joker: Folie a Deux
Movies

YouTube Video Sentiment Analysis - Data Collection / Sentiment + Emotion Analysis Script

Author: Aryan Shetty

YouTube Movie Trailer Sentiment & Emotion Analysis

1. Objective

The goal is to **analyze public sentiment and emotional reactions to movie trailers** using YouTube comments.

---

2. Data Sources

- **YouTube API**
  Used to fetch trailer video metadata and user comments.
- **Hugging Face Pretrained Models**
  - **Sentiment Analysis**: `Pretrained Model for Sentiment Analysis`
  - **Emotion Recognition**: `Pretrained Model for Emotion Recognition`

---

3. YouTube API Setup

1. Go to Google Cloud Console
2. Create a project and enable the **YouTube Data API v3**
3. Generate an API key
4. Create a `.env` file in your project folder and add:

   ```
   YOUTUBE_API_KEY=your_actual_api_key_here
   ```
5. Load it in your Python code:

   ```
   from dotenv import load_dotenv
   import os
   ```

```
load_dotenv()
YOUTUBE_API_KEY = os.getenv("YOUTUBE_API_KEY")
```

---

4. Methodology

4.1 Data Collection

- Search for YouTube videos using keywords like `"Movie Name + Trailer"`
- Retrieve metadata (title, views, likes, comments)
- Collect top-level comments for each video

4.2 Text Preprocessing

- Lowercase conversion
- URL and punctuation removal
- Non-alphabetic filtering
- Extra whitespace removal
- Truncation to 512 tokens (for model compatibility)

4.3 Data Storage

- Processed comment data saved to:
  `movie_data/<movie_name>_comments.csv`
- Sentiment & emotion plots saved to:
  `outputs/<movie_name>_sentiment_distribution.png`
  `outputs/<movie_name>_emotion_distribution.png`

---

5. Visualizations

- **Sentiment Distribution**
  Bar plot showing number of comments in each sentiment category (Very Negative → Very Positive)

- **Top Emotions**
  Horizontal bar plot showing emotion percentages per movie

---

6. ML Models

Sentiment Analysis

- **Model**: RoBERTa
- **Trained on**: Twitter data

- **Classes**: Positive, Neutral, Negative
- **Compound Score**: `positive - negative`

Emotion Recognition

- **Model**: DistilRoBERTa
- **Trained on**: English emotion-labeled datasets
- **Classes**: Joy, Sadness, Anger, Fear, Surprise, Disgust, Neutral

---

7. Considerations

- API rate limiting handled using `time.sleep(0.5)`
- Only top-level comments are analyzed (not replies)
- Long comments are truncated at 512 tokens
- Sentiment is categorized using compound score bins
- One CSV and two visualizations are generated per movie

In [8]:
```
import pandas as pd
import datetime
import time
import re
import os
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from googleapiclient.discovery import build
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from scipy.special import softmax
# this is used to import private variables from YouTube -> you can either mo
# or create a .env with them.
from dotenv import load_dotenv

import warnings
from transformers.utils import logging

# Suppress transformer-related warnings
warnings.filterwarnings("ignore", category=UserWarning, module="transformers

# Also suppress warnings from ONNX or tokenizer loading
logging.set_verbosity_error()
```

In [33]:
```
class YouTubeDataCollector:
    def __init__(self, api_key):
        """
        initialize the YouTube API client

        parameters:
        - api_key: your YouTube API key
        """
```

```
        self.youtube = build('youtube', 'v3', developerKey=api_key)

        # initialize pretrained sentiment analysis model
        # using RoBERTa model fine-tuned for sentiment analysis on tweets
        # source: https://huggingface.co/cardiffnlp/twitter-roberta-base-sen
        self.sentiment_model_name = "cardiffnlp/twitter-roberta-base-sentime
        self.sentiment_tokenizer = AutoTokenizer.from_pretrained(self.sentim
        self.sentiment_model = AutoModelForSequenceClassification.from_pretr

        # doing the same but for emotion analysis
        # source: https://huggingface.co/j-hartmann/emotion-english-distilro
        self.emotion_model_name = "j-hartmann/emotion-english-distilroberta-
        self.emotion_tokenizer = AutoTokenizer.from_pretrained(self.emotion_
        self.emotion_model = AutoModelForSequenceClassification.from_pretrai

        # get emotion labels dynamically from the model configuration
        self.emotion_labels = [
            self.emotion_model.config.id2label[i]
            for i in range(len(self.emotion_model.config.id2label))
        ]

        # create emotion column names
        self.emotion_columns = [f"{label.lower()}_emotion" for label in self

        # create folder for data if it doesn't exist
        if not os.path.exists('movie_data'):
            os.makedirs('movie_data')

    def clean_text(self, text):
        """
        clean and preprocess text data

        parameters:
        - text: text to clean

        returns:
        - cleaned text
        """
        if text is None:
            return ""

        # to lowercase
        text = text.lower()

        # remove urls
        text = re.sub(r'http\S+', '', text)

        # keep only letters and spaces
        text = re.sub(r'[^a-zA-Z\s]', '', text)

        # remove extra whitespace
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def get_sentiment_scores(self, text):
```

```
        """
        calculate sentiment scores for a given text using the pretrained moc

        parameters:
        - text: text to analyze

        returns:
        - dictionary with sentiment scores
        """
        if not text:
            # return neutral sentiment if text is empty
            return {
                'compound': 0,
                'pos': 0,
                'neu': 1,
                'neg': 0
            }

        # truncate text if it's too long for the model
        max_length = 512
        if len(text) > max_length:
            text = text[:max_length]

        encoded_input = self.sentiment_tokenizer(text, return_tensors='pt',

        # get model output
        with torch.no_grad():
            output = self.sentiment_model(**encoded_input)

        # get probabilities
        scores = softmax(output.logits[0].numpy())

        # map scores to sentiment categories (negative, neutral, positive)
        sentiment_scores = {
            'neg': float(scores[0]),
            'neu': float(scores[1]),
            'pos': float(scores[2]),
            'compound': float(scores[2] - scores[0])
        }

        return sentiment_scores

    def get_emotion_scores(self, text):
        """
        calculate emotion scores

        parameters:
        - text: text to analyze

        returns:
        - dictionary with emotion scores
        """
        if not text:
            # return neutral emotion scores if text is empty
            return {label.lower(): 0 for label in self.emotion_labels}
```

```
        # truncate text if it's too long for the model
        max_length = 512
        if len(text) > max_length:
            text = text[:max_length]

        encoded_input = self.emotion_tokenizer(text, return_tensors='pt', tr

        # get model output
        with torch.no_grad():
            output = self.emotion_model(**encoded_input)

        # get probabilities
        scores = softmax(output.logits[0].numpy())

        # map scores to emotion categories dynamically
        emotion_mapping = {
            self.emotion_model.config.id2label[i].lower(): float(scores[i])
            for i in range(len(self.emotion_labels))
        }

        return emotion_mapping

    def search_videos(self, search_term, max_results=10):
        """
        search for videos related to a given term

        parameters:
        - search_term: term to search for
        - max_results: maximum number of videos to retrieve

        returns:
        - list of video IDs
        """
        search_response = self.youtube.search().list(
            q=search_term,
            part='id',
            maxResults=max_results,
            type='video',
            relevanceLanguage='en',
            order='relevance'
        ).execute()

        video_ids = [item['id']['videoId'] for item in search_response.get('
        return video_ids


    def get_video_details(self, video_ids):
        """
        get details for given video IDs

        parameters:
        - video_ids: list of video IDs

        returns:
        - dict with video details
        """
```

```
        video_response = self.youtube.videos().list(
            id=','.join(video_ids),
            part='snippet,statistics'
        ).execute()

        videos = {}
        for item in video_response.get('items', []):
            video_id = item['id']
            snippet = item['snippet']
            statistics = item['statistics']

            videos[video_id] = {
                'title': snippet.get('title', ''),
                'channel': snippet.get('channelTitle', ''),
                'published_at': snippet.get('publishedAt', ''),
                'view_count': int(statistics.get('viewCount', 0)),
                'like_count': int(statistics.get('likeCount', 0)) if 'likeCo
                'comment_count': int(statistics.get('commentCount', 0)) if '
            }

        return videos


    def get_video_comments(self, video_id, max_comments=100):
        """
        get comments for a specific video

        parameters:
        - video_id: ID of the video
        - max_comments: maximum number of comments to retrieve

        returns:
        - list of comments
        """
        comments = []
        next_page_token = None

        try:
            while len(comments) < max_comments:
                # request comment threads
                request = self.youtube.commentThreads().list(
                    part='snippet',
                    videoId=video_id,
                    maxResults=min(100, max_comments - len(comments)),
                    pageToken=next_page_token,
                    textFormat='plainText'
                )

                response = request.execute()

                # process comment threads
                for item in response['items']:
                    comment = item['snippet']['topLevelComment']['snippet']

                    # get comment data
                    comment_data = {
```

```
                        'id': item['id'],
                        'text': comment['textDisplay'],
                        'author': comment['authorDisplayName'],
                        'published_at': comment['publishedAt'],
                        'like_count': comment['likeCount'],
                        'video_id': video_id
                    }

                    comments.append(comment_data)

                # check for next page
                next_page_token = response.get('nextPageToken')
                if not next_page_token or len(comments) >= max_comments:
                    break

                # sleep to avoid rate limits
                time.sleep(0.5)

        except Exception as e:
            print(f"Error collecting comments for video {video_id}: {e}")

        return comments

    def collect_youtube_data(self, search_term, max_videos=5, max_comments_p
        """
        collect YouTube data for a specific search term

        parameters:
        - search_term: term to search for
        - max_videos: maximum number of videos to analyze
        - max_comments_per_video: maximum number of comments to retrieve per

        returns:
        - df with collected data
        """
        # search for videos
        video_ids = self.search_videos(search_term, max_results=max_videos)

        # get video details
        videos = self.get_video_details(video_ids)

        # collect comments for each video
        all_comments = []
        total_comments = 0

        for video_id, video_info in videos.items():
            print(f"Collecting comments for video: {video_info['title']}")

            # get comments
            video_comments = self.get_video_comments(video_id, max_comments=

            # process each comment
            for comment in video_comments:
                # clean comment text
                clean_comment = self.clean_text(comment['text'])
```

```
                # get sentiment scores
                sentiment_scores = self.get_sentiment_scores(clean_comment)

                # get emotion scores
                emotion_scores = self.get_emotion_scores(clean_comment)

                # process comment
                comment_data = {
                    'id': comment['id'],
                    'video_id': video_id,
                    'video_title': video_info['title'],
                    'channel': video_info['channel'],
                    'text': comment['text'],
                    'author': comment['author'],
                    'published_at': comment['published_at'],
                    'like_count': comment['like_count'],
                    'compound_sentiment': sentiment_scores['compound'],
                    'positive_sentiment': sentiment_scores['pos'],
                    'neutral_sentiment': sentiment_scores['neu'],
                    'negative_sentiment': sentiment_scores['neg'],
                    **{f"{k}_emotion": v for k, v in emotion_scores.items()}
                }

                all_comments.append(comment_data)

            total_comments += len(video_comments)

            # sleep to avoid rate limits
            time.sleep(0.5)

        # create dataframe
        comments_df = pd.DataFrame(all_comments)

        # save to CSV
        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        search_term_cleaned = search_term.replace(" ", "_").lower()

        filename = f"movie_data/{search_term_cleaned}_youtube_comments_{time

        comments_df.to_csv(filename, index=False)

        print(f"Data collection is complete: {len(videos)} videos and {total
        print(f"Data saved to {filename}")

        return comments_df

    def analyze_sentiment_and_emotion_distribution(self, data):
        """
        analyze the sentiment distribution of collected data

        parameters:
        - data: df with collected data

        returns:
        - dict with sentiment distribution analysis
        """
```

```
        # calculate averages for sentiment scores
        sentiment_avg = {
            'Average compound score': data['compound_sentiment'].mean(),
            'Average positive score': data['positive_sentiment'].mean(),
            'Average neutral score': data['neutral_sentiment'].mean(),
            'Average negative score': data['negative_sentiment'].mean()
        }

        # categorize sentiments
        data['sentiment_category'] = pd.cut(
            data['compound_sentiment'],
            bins=[-2, -0.6, -0.2, 0.2, 0.6, 2],
            labels=['Very negative', 'Negative', 'Neutral', 'Positive', 'Ver
        )

        # count by category
        sentiment_counts = data['sentiment_category'].value_counts().to_dict

        # calculate percentages
        total = sum(sentiment_counts.values())
        sentiment_percentages = {k: (v / total) * 100 for k, v in sentiment_

        # emotion analysis
        emotion_columns = [col for col in data.columns if col.endswith('_emo

        # calculate average emotion scores
        emotion_avg = {f'Average {col.split("_")[0]} emotion': data[col].mea

        # calculate total emotion scores across all content
        total_emotion_scores = {}
        for col in emotion_columns:
            emotion_name = col.split('_')[0]
            total_emotion_scores[emotion_name] = data[col].sum()

        # sort emotions
        sorted_emotions = sorted(total_emotion_scores.items(), key=lambda x:

        emotions_dict = {
            'emotions': [emotion for emotion, score in sorted_emotions],
            'emotions_scores': {emotion: score for emotion, score in sorted_
        }

        # calculate percentages for emotions
        total_emotion_score = sum(total_emotion_scores.values())
        emotions_percentages = {
            emotion: (score / total_emotion_score) * 100
            for emotion, score in emotions_dict['emotions_scores'].items()
        }
        emotions_dict['emotions_percentages'] = emotions_percentages

        # combine results
        analysis_result = {
            'sentiment_avg': sentiment_avg,
            'sentiment_counts': sentiment_counts,
            'sentiment_percentages': sentiment_percentages,
            'emotion_avg': emotion_avg,
```

```
            'emotions': emotions_dict
        }

        return analysis_result

    def analyze_by_video(self, data):
        """
        analyze sentiment and emotion by video

        parameters:
        - data: df with collected data

        returns:
        - df with analysis by video
        """
        # get unique videos
        videos = data['video_id'].unique()

        video_analysis = []

        for video_id in videos:
            # filter data for this video
            video_data = data[data['video_id'] == video_id]

            # get video info
            video_title = video_data['video_title'].iloc[0]
            channel = video_data['channel'].iloc[0]

            # calculate average sentiment
            avg_sentiment = video_data['compound_sentiment'].mean()
            avg_positive = video_data['positive_sentiment'].mean()
            avg_negative = video_data['negative_sentiment'].mean()

            # calculate dominant emotion
            emotion_columns = [col for col in data.columns if col.endswith('
            emotion_averages = {col.split('_')[0]: video_data[col].mean() fo
            dominant_emotion = max(emotion_averages.items(), key=lambda x: x

            # store analysis
            video_analysis.append({
                'video_id': video_id,
                'video_title': video_title,
                'channel': channel,
                'comment_count': len(video_data),
                'avg_sentiment': avg_sentiment,
                'avg_positive': avg_positive,
                'avg_negative': avg_negative,
                'dominant_emotion': dominant_emotion
            })

        return pd.DataFrame(video_analysis)
```

In [34]:
```
# load credentials from .env file
load_dotenv()
YOUTUBE_API_KEY = os.getenv("YOUTUBE_API_KEY")
```

```
# initialize collector with YouTube API key
collector = YouTubeDataCollector(YOUTUBE_API_KEY)

movies = ["The Flash", "Barbie", "Oppenheimer", "Snow White"]

for movie in movies:
    print(f"\nProcessing: {movie}")
    search_term = movie + " trailer"
    cleaned_name = movie.lower().replace(" ", "_")

    # collect data
    data = collector.collect_youtube_data(
        search_term=search_term,
        max_videos=3,
        max_comments_per_video=100
    )

    # save raw data
    data.to_csv(f"movie_data/{cleaned_name}_comments.csv", index=False)

    # analyze sentiment and emotion
    analysis = collector.analyze_sentiment_and_emotion_distribution(data)

    # aentiment Output
    print(f"\nSentiment analysis results for {movie}:")
    print("Average sentiment scores:")
    for k, v in analysis['sentiment_avg'].items():
        print(f"  {k}: {v:.4f}")

    print("\nSentiment distribution:")
    for k, v in analysis['sentiment_counts'].items():
        percentage = analysis['sentiment_percentages'][k]
        print(f"  {k}: {v} ({percentage:.2f}%)")

    # Emotion Output
    print("\nEmotion analysis results:")
    print("Average emotion scores:")
    for k, v in analysis['emotion_avg'].items():
        print(f"  {k}: {v:.4f}")

    print("\nTop emotions:")
    emotions_data = analysis['emotions']
    for emotion, score in zip(
        emotions_data['emotions'],
        emotions_data['emotions_scores'].values()
    ):
        percentage = emotions_data['emotions_percentages'][emotion]
        print(f"  {emotion.capitalize()}: {score:.2f} ({percentage:.2f}%)")

    print("\n" + "-"*60 + "\n")
```

```
Processing: The Flash
Collecting comments for video: The Flash – Official Trailer
Collecting comments for video: The Flash - Official Trailer 2
Collecting comments for video: THE FLASH - FINAL TRAILER
Data collection is complete: 3 videos and 300 comments
Data saved to movie_data/the_flash_trailer_youtube_comments_20250412_231618.
csv

Sentiment analysis results for The Flash:
Average sentiment scores:
  Average compound score: 0.1020
  Average positive score: 0.3308
  Average neutral score: 0.4405
  Average negative score: 0.2287

Sentiment distribution:
  Neutral: 113 (37.67%)
  Very positive: 80 (26.67%)
  Very negative: 48 (16.00%)
  Negative: 35 (11.67%)
  Positive: 24 (8.00%)

Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.0455
  Average disgust emotion: 0.0296
  Average fear emotion: 0.0491
  Average joy emotion: 0.1971
  Average neutral emotion: 0.2763
  Average sadness emotion: 0.1112
  Average surprise emotion: 0.1977

Top emotions:
  Neutral: 82.90 (30.48%)
  Surprise: 59.30 (21.80%)
  Joy: 59.14 (21.74%)
  Sadness: 33.37 (12.27%)
  Fear: 14.74 (5.42%)
  Anger: 13.66 (5.02%)
  Disgust: 8.89 (3.27%)

------------------------------------------------------------


Processing: Barbie
Collecting comments for video: Barbie | Main Trailer
Collecting comments for video: Barbie | Teaser Trailer
Collecting comments for video: Barbie | Teaser Trailer 2
Data collection is complete: 3 videos and 300 comments
Data saved to movie_data/barbie_trailer_youtube_comments_20250412_231641.csv

Sentiment analysis results for Barbie:
Average sentiment scores:
  Average compound score: 0.0406
  Average positive score: 0.2454
  Average neutral score: 0.5498
```

```
  Average negative score: 0.2048

Sentiment distribution:
  Neutral: 157 (52.33%)
  Very positive: 50 (16.67%)
  Very negative: 46 (15.33%)
  Positive: 29 (9.67%)
  Negative: 18 (6.00%)

Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.0448
  Average disgust emotion: 0.0442
  Average fear emotion: 0.0419
  Average joy emotion: 0.1672
  Average neutral emotion: 0.3217
  Average sadness emotion: 0.1018
  Average surprise emotion: 0.1250

Top emotions:
  Neutral: 96.52 (38.00%)
  Joy: 50.15 (19.75%)
  Surprise: 37.51 (14.77%)
  Sadness: 30.55 (12.03%)
  Anger: 13.44 (5.29%)
  Disgust: 13.25 (5.22%)
  Fear: 12.58 (4.95%)

------------------------------------------------------------


Processing: Oppenheimer
Collecting comments for video: Oppenheimer | New Trailer
Collecting comments for video: Oppenheimer | Official Trailer
Collecting comments for video: Oppenheimer First Look (2023)
Data collection is complete: 3 videos and 300 comments
Data saved to movie_data/oppenheimer_trailer_youtube_comments_20250412_23170
9.csv

Sentiment analysis results for Oppenheimer:
Average sentiment scores:
  Average compound score: 0.0750
  Average positive score: 0.3279
  Average neutral score: 0.4193
  Average negative score: 0.2528

Sentiment distribution:
  Neutral: 94 (31.33%)
  Very positive: 73 (24.33%)
  Very negative: 58 (19.33%)
  Positive: 44 (14.67%)
  Negative: 31 (10.33%)

Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.0698
```

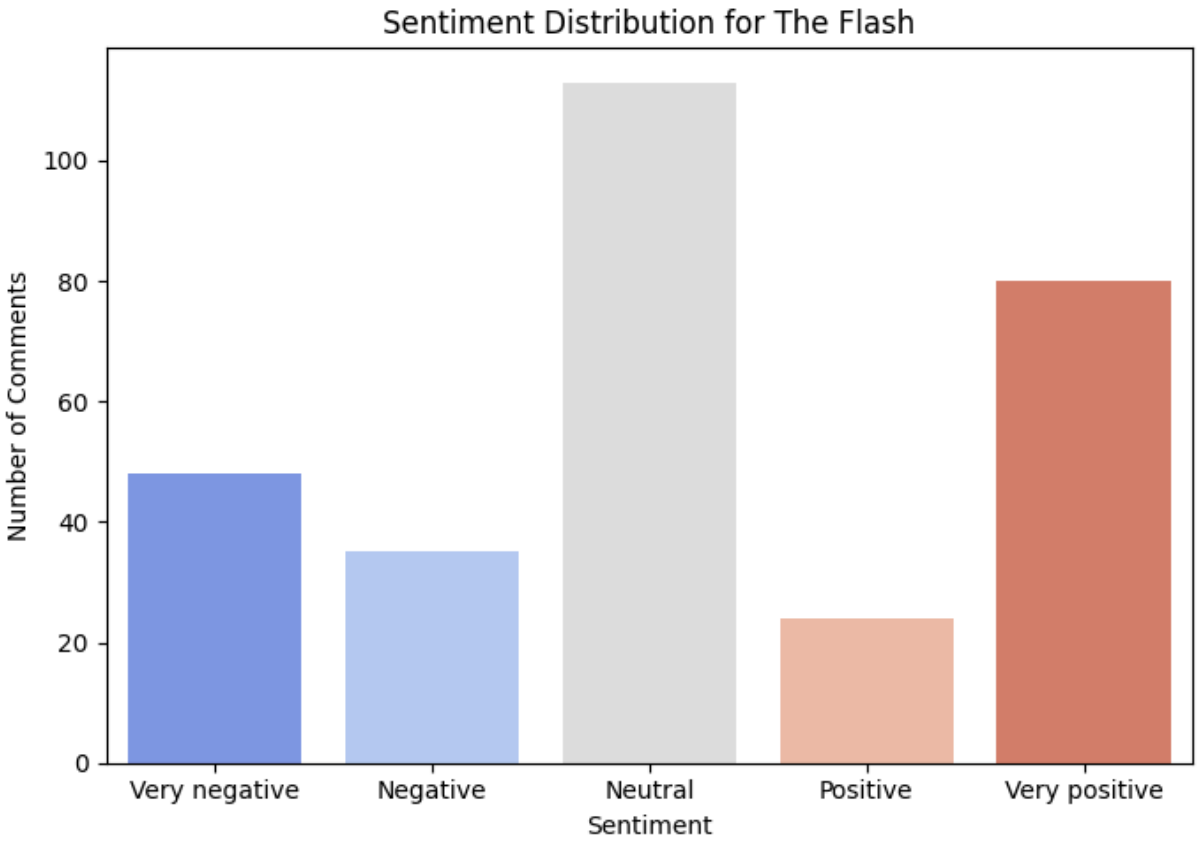
```
  Average disgust emotion: 0.0248
  Average fear emotion: 0.0545
  Average joy emotion: 0.2202
  Average neutral emotion: 0.2393
  Average sadness emotion: 0.1353
  Average surprise emotion: 0.1929

Top emotions:
  Neutral: 71.80 (25.55%)
  Joy: 66.05 (23.51%)
  Surprise: 57.86 (20.59%)
  Sadness: 40.58 (14.44%)
  Anger: 20.94 (7.45%)
  Fear: 16.34 (5.82%)
  Disgust: 7.43 (2.65%)

------------------------------------------------------------


Processing: Snow White
Collecting comments for video: Disney’s Snow White | Official Trailer | In T
heaters March 21
Collecting comments for video: Disney’s Snow White | Teaser Trailer | In The
aters March 21
Collecting comments for video: Disney’s Snow White | Official Trailer | In c
inemas March 2025
Data collection is complete: 3 videos and 300 comments
Data saved to movie_data/snow_white_trailer_youtube_comments_20250412_23173
4.csv

Sentiment analysis results for Snow White:
Average sentiment scores:
  Average compound score: -0.2760
  Average positive score: 0.1701
  Average neutral score: 0.3838
  Average negative score: 0.4461

Sentiment distribution:
  Very negative: 121 (40.33%)
  Neutral: 82 (27.33%)
  Negative: 48 (16.00%)
  Very positive: 33 (11.00%)
  Positive: 16 (5.33%)

Emotion analysis results:
Average emotion scores:
  Average anger emotion: 0.1065
  Average disgust emotion: 0.0592
  Average fear emotion: 0.0529
  Average joy emotion: 0.1311
  Average neutral emotion: 0.2571
  Average sadness emotion: 0.1644
  Average surprise emotion: 0.1922

Top emotions:
  Neutral: 77.12 (26.69%)
```

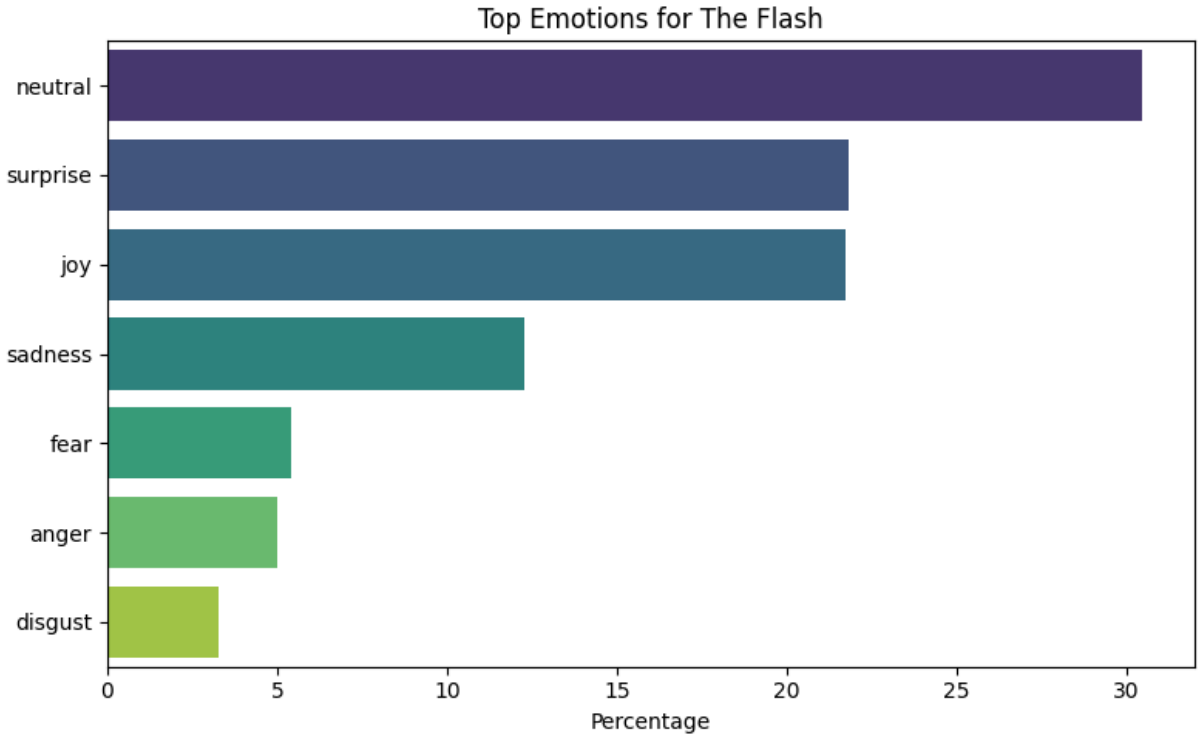
```
Surprise: 57.67 (19.96%)
Sadness: 49.32 (17.07%)
Joy: 39.32 (13.60%)
Anger: 31.94 (11.05%)
Disgust: 17.75 (6.14%)
Fear: 15.88 (5.50%)

------------------------------------------------------------
```

In [5]:
```
def plot_sentiment_distribution(data, movie_title):
    plt.figure(figsize=(7, 5))
    sns.countplot(x='sentiment_category', data=data, hue='sentiment_category
    plt.title(f"Sentiment Distribution for {movie_title}")
    plt.xlabel("Sentiment")
    plt.ylabel("Number of Comments")
    plt.tight_layout()
    plt.savefig(f"outputs/{movie_title.lower().replace(' ', '_')}_sentiment_
    plt.show()

def plot_top_emotions(analysis, movie_title):
    emotions = analysis['emotions']['emotions']
    percentages = analysis['emotions']['emotions_percentages']

    plt.figure(figsize=(8, 5))
    sns.barplot(x=list(percentages.values()), y=emotions, hue=emotions, lege
    plt.xlabel("Percentage")
    plt.title(f"Top Emotions for {movie_title}")
    plt.tight_layout()
    plt.savefig(f"outputs/{movie_title.lower().replace(' ', '_')}_emotion_di
    plt.show()
```

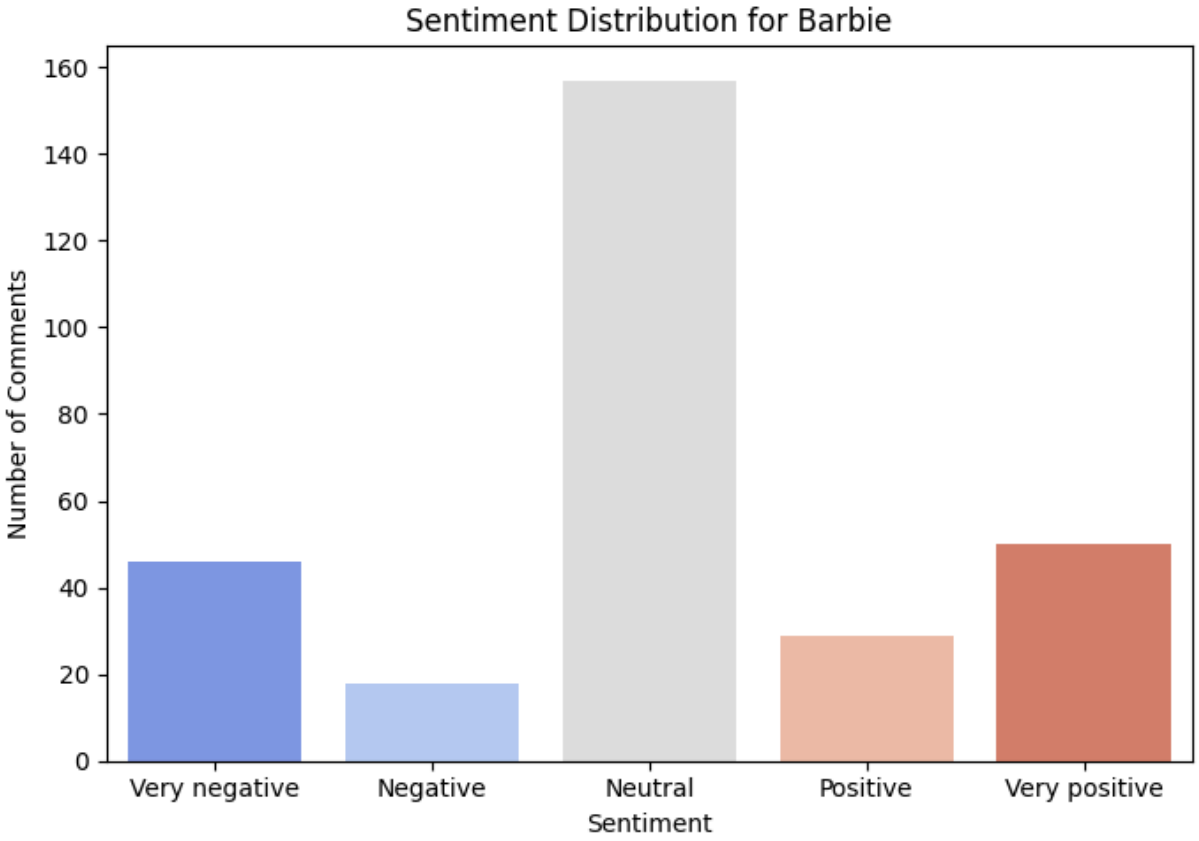
In [6]:
```
os.makedirs("outputs", exist_ok=True)

for movie in movies:
    cleaned_name = movie.lower().replace(" ", "_")
    filename = f"movie_data/{cleaned_name}_comments.csv"

    try:
        print(f"\nGenerating plots for: {movie}")
        data = pd.read_csv(filename)

        # run analysis and generate plots
        analysis = collector.analyze_sentiment_and_emotion_distribution(data
        plot_sentiment_distribution(data, movie)
        plot_top_emotions(analysis, movie)

    except Exception as e:
        print(f"Failed for {movie}: {e}")
```

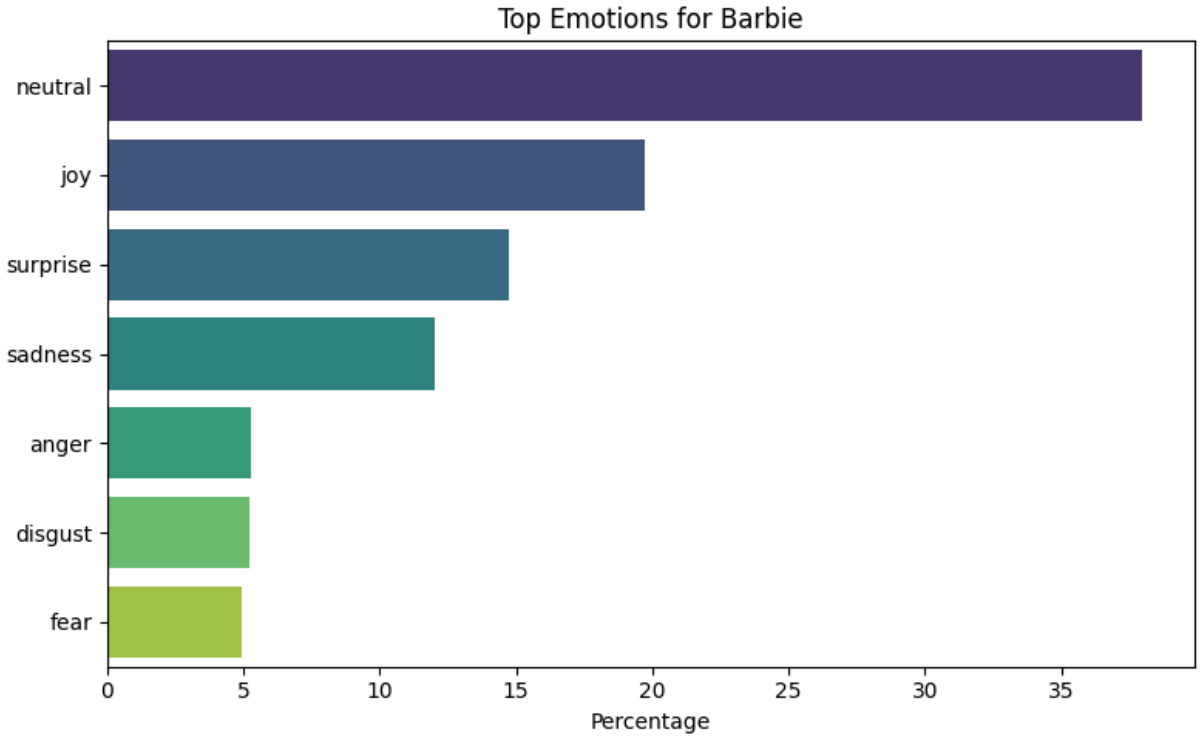
```
Generating plots for: The Flash
```



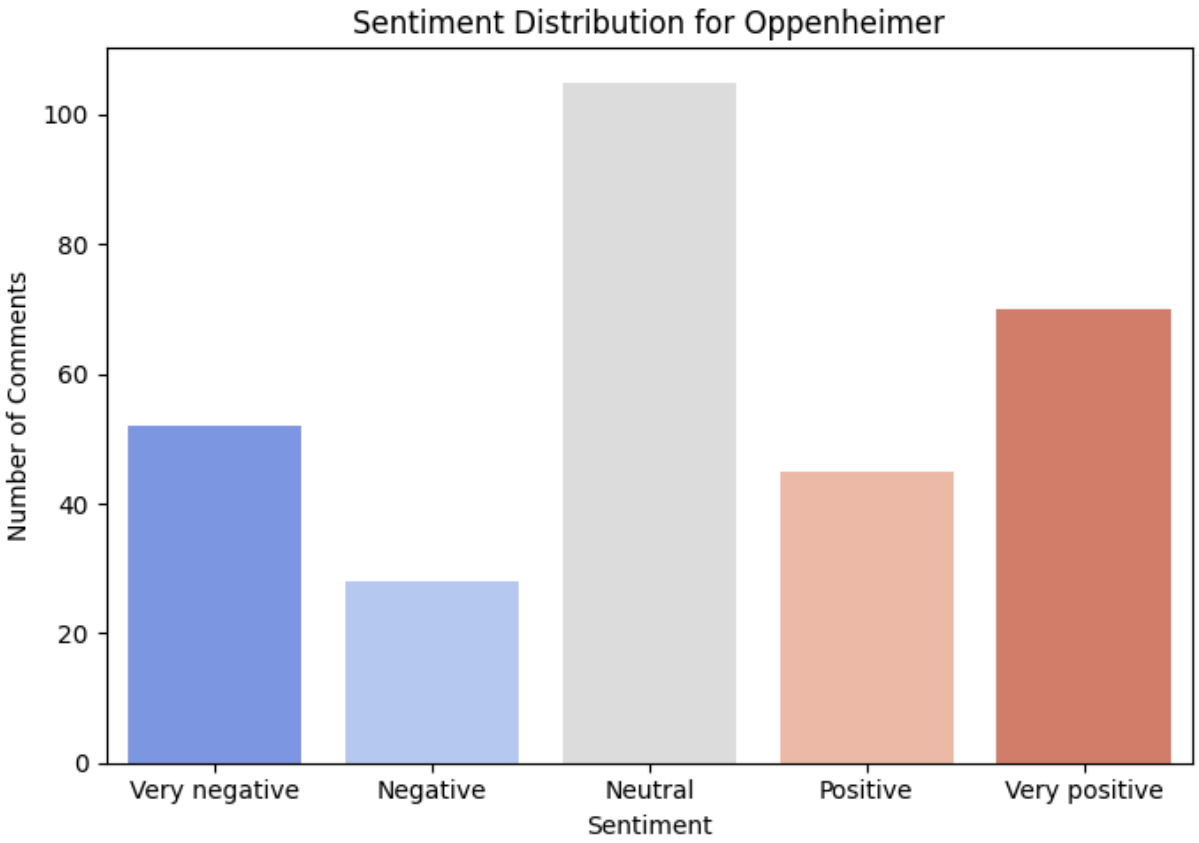


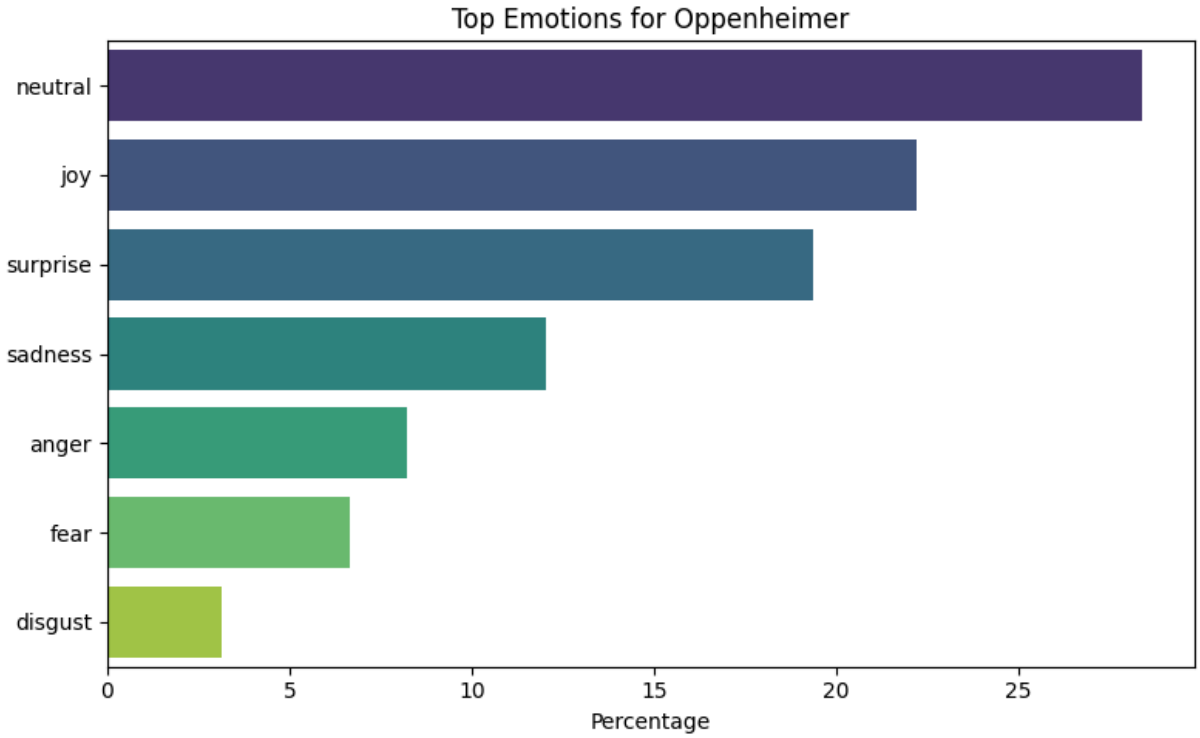

```
Generating plots for: Barbie
```



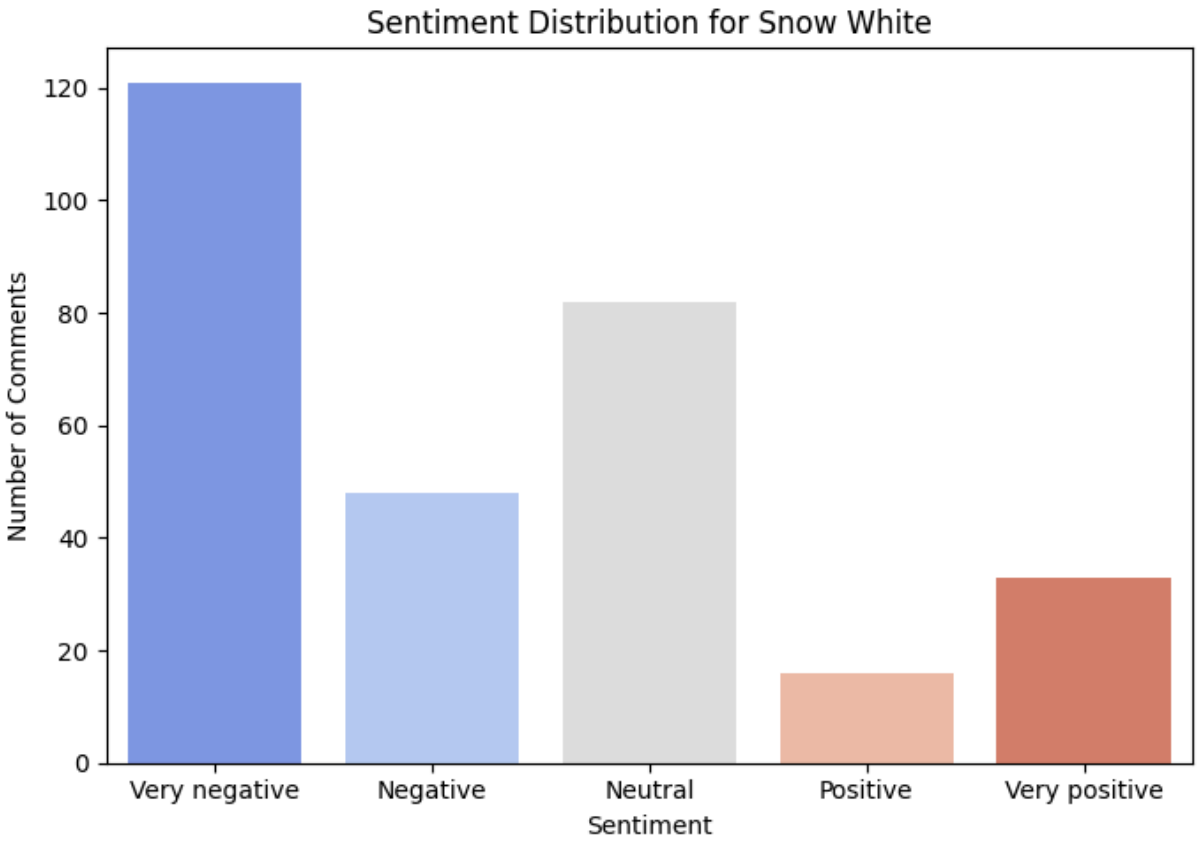


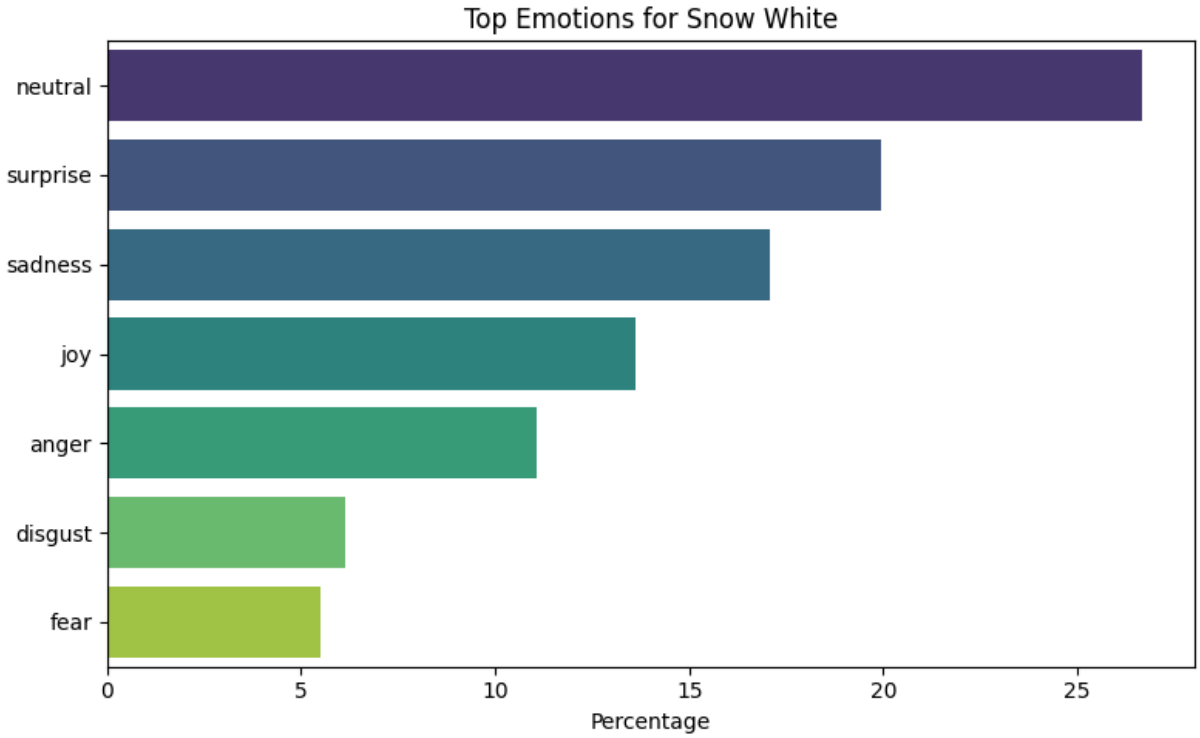

```
Generating plots for: Oppenheimer
```






```
Generating plots for: Snow White
```


Sentiment Distribution for Snow White
Number of Comments
0
20
40
60
80
100
120
Very negative
Negative
Neutral
Positive
Very positive
Sentiment


Top Emotions for Snow White
neutral
surprise
sadness
joy
anger
disgust
fear
0
5
10
15
20
25
Percentage

ML Model Prediction

Author: Niharika belavadi Shekar

Workflow Summary

Step 1: Data Cleaning & Preprocessing

- Removed missing/invalid rows
- Normalized column names
- Converted `release_date` to datetime
- Added features like `release_year`, `release_month`, `cast_count`, `genre_count`, `keyword_count`
- Extracted top 50 directors and production companies; encoded others as 'other'
- One-hot encoded categorical columns for XGBoost and LightGBM
- For CatBoost, used categorical column support directly

Step 2: Baseline Modeling (Before Sentiment)

We trained 3 models using only structured features (metadata):

Model 1: XGBoost

- Validation Accuracy (Custom ±10M): 65.23%
- Test Accuracy (Custom ±10M): 64.78%

Model 2: CatBoost

- Validation Accuracy: 65.05%
- Test Accuracy: 64.94%

Model 3: LightGBM

- Validation Accuracy: 65.42%
- Test Accuracy: 65.68%

Step 3: Sentiment & Emotion Signal Integration

We extracted public sentiment and emotional reactions using ML models. Due to computational constraints, instead of fetching data for all movies from Reddit and YouTube APIs, we generated sentiment and emotion scores using pretrained models.

Added 9 new features:

- Sentiment: `very_negative`, `neutral`, `very_positive`
- Emotion: `anger`, `disgust`, `joy`, `sadness`, etc.
- These columns were given 3x weight during model training

Step 4: Model Re-training (After Sentiment Enrichment)

Model 1: XGBoost

- Validation Accuracy: 84.86%
- Test Accuracy: 85.12%

Model 2: CatBoost

- Validation Accuracy: 88.54%
- Test Accuracy: 89.30%

Model 3: LightGBM

- Validation Accuracy: 78.63%
- Test Accuracy: 78.71%

Key Insights

- Sentiment and emotion scores significantly improved prediction accuracy.
- CatBoost emerged as the top performer after enrichment.
- Weighting social features helped the model better understand audience anticipation and emotion.

Future Work

- Automate real-time data fetching via Reddit/YouTube APIs
- Perform time-series analysis of sentiment evolution before and after release
- Try deep learning with attention over time-stamped comments
- Deploy as an interactive dashboard or web app

Tech Stack

- Python, Pandas, NumPy
- XGBoost, CatBoost, LightGBM
- HuggingFace Transformers (DistilBERT)
- Text2Emotion for emotion scoring

- Matplotlib for visualization

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [17]:
```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/tmdb_enriched_movies.csv')
```

In [18]:
```
import pandas as pd

# Loading CSV
# df = pd.read_csv('tmdb_enriched_movies.csv')

# Cleaning column names
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')

# Converting data types
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
num_cols = ['budget', 'revenue', 'runtime', 'vote_average', 'vote_count']
for col in num_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Dropping rows with missing budget or revenue
df.dropna(subset=['budget', 'revenue'], inplace=True)

# Stripping and lowercasing text fields
text_cols = ['language', 'genres', 'production_companies', 'top_cast', 'keyw
for col in text_cols:
    df[col] = df[col].astype(str).str.strip().str.lower()

# Splitting multi-value columns
df['genres_list'] = df['genres'].str.split(', ')
df['top_cast_list'] = df['top_cast'].str.split(', ')
df['keywords_list'] = df['keywords'].str.split(', ')

# Removing duplicate entries
df.drop_duplicates(subset=['movie_id', 'title'], inplace=True)

# Filtering out invalid rows
df = df[(df['budget'] >= 0) & (df['revenue'] >= 0) & (df['runtime'] > 0)]

# (TBD) Removing unwanted columns

# Previewing final dataset
df.head()
```

Out[18]:

	movie_id	imdb_id	title	vote_average	vote_count	status	release_date
0	27205	tt1375666	Inception	8.369	37309	released	2010-07-15
1	157336	tt0816692	Interstellar	8.453	36903	released	2014-11-05
2	155	tt0468569	The Dark Knight	8.519	33688	released	2008-07-16
3	19995	tt0499549	Avatar	7.588	32126	released	2009-12-15
4	24428	tt0848228	The Avengers	7.735	31521	released	2012-04-25

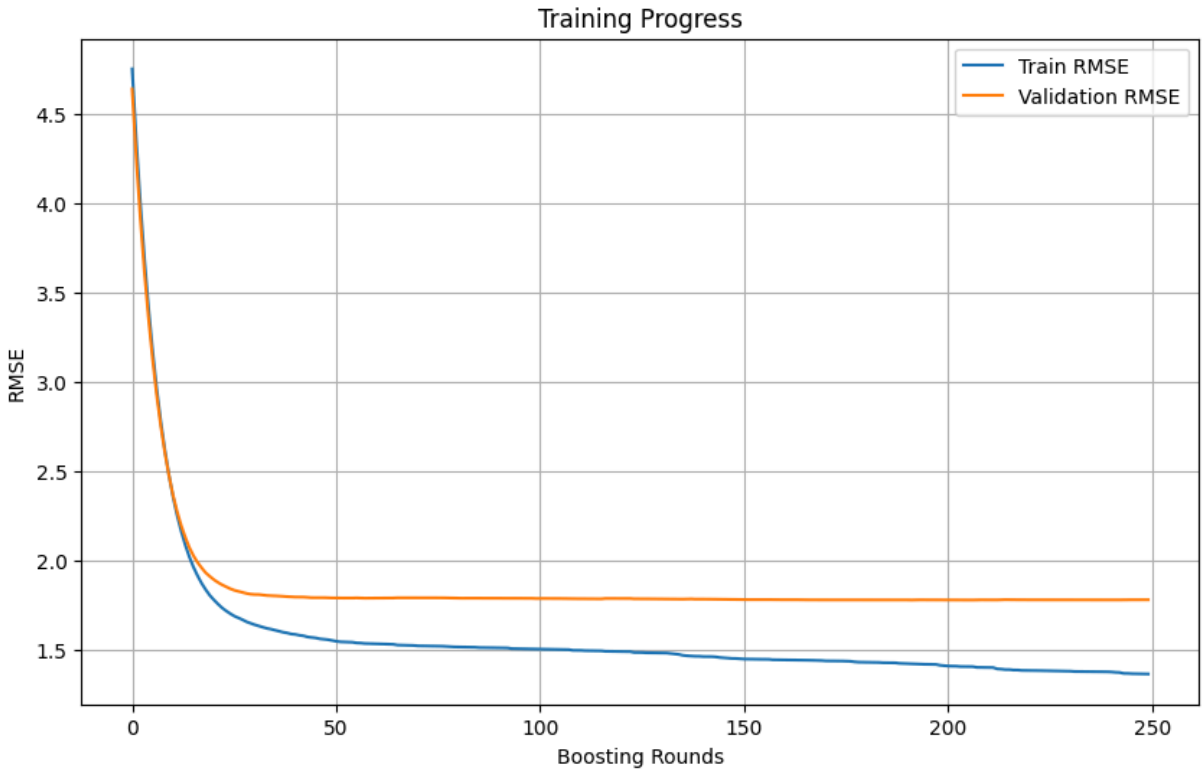
In [19]:
```
# Saving the cleaned dataset to a new CSV file
df.to_csv('/content/drive/My Drive/cleaned_movies_dataset.csv', index=False)
```

In [20]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor
import matplotlib.pyplot as plt

# === 1. Loading cleaned dataset ===
df = pd.read_csv('/content/drive/My Drive/cleaned_movies_dataset.csv')

# === 2. Converting release_date column to datetime format ===
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# === 3. Performing feature engineering ===
df['release_year'] = df['release_date'].dt.year
df['release_month'] = df['release_date'].dt.month
df['cast_count'] = df['top_cast'].fillna('').str.split(', ').apply(len)
df['genre_count'] = df['genres'].fillna('').str.split(', ').apply(len)
```

```
df['keyword_count'] = df['keywords'].fillna('').str.split(', ').apply(len)

# Extracting top 50 directors and encoding others as 'other'
top_directors = df['director'].value_counts().nlargest(50).index
df['director_clean'] = df['director'].apply(lambda x: x if x in top_director

# Extracting top 50 production companies and encoding others as 'other'
df['main_prod_company'] = df['production_companies'].str.split(', ').str[0]
top_companies = df['main_prod_company'].value_counts().nlargest(50).index
df['main_prod_company'] = df['main_prod_company'].apply(lambda x: x if x in

# Applying log transformation to the revenue column
df['log_revenue'] = np.log1p(df['revenue'])

# === 4. Applying one-hot encoding to categorical features ===
categorical_cols = ['language', 'status', 'main_prod_company', 'director_cle
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# === 5. Defining features and target column ===
drop_cols = ['movie_id', 'imdb_id', 'title', 'revenue', 'release_date',
             'genres', 'top_cast', 'keywords', 'production_companies',
             'director', 'genres_list', 'top_cast_list', 'keywords_list']

features = [col for col in df.columns if col not in drop_cols + ['log_revenu
X = df[features]
y = df['log_revenue']

# === 6. Splitting dataset into training, validation, and test sets ===
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# === 7. Initializing the XGBoost model with tracking and early stopping ===
model = XGBRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    n_jobs=-1,
    objective='reg:squarederror',
    eval_metric='rmse',
    early_stopping_rounds=50
)

evals = [(X_train, y_train), (X_val, y_val)]

# === 8. Fitting the model and monitoring progress ===
model.fit(
    X_train, y_train,
    eval_set=evals,
    verbose=100
)

# === 9. Defining evaluation function with custom accuracy ===
def evaluate(model, X, y_true, label):
    y_pred = model.predict(X)
    y_pred_rev = np.expm1(y_pred)
```

```
    y_true_rev = np.expm1(y_true)

    mae = mean_absolute_error(y_true_rev, y_pred_rev)
    rmse = np.sqrt(mean_squared_error(y_true_rev, y_pred_rev))
    within_range = np.abs(y_pred_rev - y_true_rev) <= 10000000
    custom_acc = np.mean(within_range)

    print(f"\n📊 {label} Results:")
    print(f"MAE: {mae:,.2f}")
    print(f"RMSE: {rmse:,.2f}")
    print(f"Custom ±10M Accuracy: {custom_acc * 100:.2f}%")

# === 10. Plotting training progress ===
def plot_training_progress(model):
    evals_result = model.evals_result()
    plt.figure(figsize=(10, 6))
    plt.plot(evals_result['validation_0']['rmse'], label='Train RMSE')
    plt.plot(evals_result['validation_1']['rmse'], label='Validation RMSE')
    plt.xlabel('Boosting Rounds')
    plt.ylabel('RMSE')
    plt.title('Training Progress')
    plt.legend()
    plt.grid(True)
    plt.show()

# === 11. Evaluating model performance ===
evaluate(model, X_val, y_val, "Validation")
evaluate(model, X_test, y_test, "Test")

# === 12. Displaying RMSE plot over boosting rounds ===
plot_training_progress(model)
```

```
[0]     validation_0-rmse:4.74789       validation_1-rmse:4.63617
[100]   validation_0-rmse:1.50660       validation_1-rmse:1.78955
[200]   validation_0-rmse:1.41188       validation_1-rmse:1.78128
[249]   validation_0-rmse:1.36740       validation_1-rmse:1.78220

📊 Validation Results:
MAE: 23,493,934.23
RMSE: 73,459,080.97
Custom ±10M Accuracy: 65.23%

📊 Test Results:
MAE: 23,461,026.11
RMSE: 63,021,622.95
Custom ±10M Accuracy: 64.78%
```


Training Progress
Train RMSE
Validation RMSE
RMSE
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0
50
100
150
200
250
Boosting Rounds

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/tmdb_enriched_movies.csv')
```

In [ ]:
```
import pandas as pd

# Loading CSV
# df = pd.read_csv('tmdb_enriched_movies.csv')

# Cleaning column names
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')

# Converting data types
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
num_cols = ['budget', 'revenue', 'runtime', 'vote_average', 'vote_count']
for col in num_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Dropping rows with missing budget or revenue
df.dropna(subset=['budget', 'revenue'], inplace=True)

# Stripping and lowercasing text fields
text_cols = ['language', 'genres', 'production_companies', 'top_cast', 'keyw
for col in text_cols:
    df[col] = df[col].astype(str).str.strip().str.lower()

# Splitting multi-value columns
df['genres_list'] = df['genres'].str.split(', ')
df['top_cast_list'] = df['top_cast'].str.split(', ')
df['keywords_list'] = df['keywords'].str.split(', ')

# Removing duplicate entries
df.drop_duplicates(subset=['movie_id', 'title'], inplace=True)

# Filtering out invalid rows
df = df[(df['budget'] >= 0) & (df['revenue'] >= 0) & (df['runtime'] > 0)]

# (TBD) Removing unwanted columns

# Previewing final dataset
df.head()
```

Out[ ]:

	movie_id	imdb_id	title	vote_average	vote_count	status	release_date
0	27205	tt1375666	Inception	8.369	37309	released	2010-07-15
1	157336	tt0816692	Interstellar	8.453	36903	released	2014-11-05
2	155	tt0468569	The Dark Knight	8.519	33688	released	2008-07-16
3	19995	tt0499549	Avatar	7.588	32126	released	2009-12-15
4	24428	tt0848228	The Avengers	7.735	31521	released	2012-04-25

In [4]:
```
# Saving the cleaned dataset to a new CSV file
df.to_csv('/content/drive/My Drive/cleaned_movies_dataset.csv', index=False)
```

In [ ]:
```
# Run only once
!pip install numpy==1.23.5 --force-reinstall
!pip install catboost --force-reinstall
```

```
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
Collecting numpy==1.23.5
  Using cached numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (2.3 kB)
Using cached numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (17.1 MB)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
Installing collected packages: numpy
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.2.3 which i
s incompatible.
pymc 5.21.2 requires numpy>=1.25.0, but you have numpy 1.23.5 which is incom
patible.
chex 0.1.89 requires numpy>=1.24.1, but you have numpy 1.23.5 which is incom
patible.
treescope 0.1.9 requires numpy>=1.25.2, but you have numpy 1.23.5 which is i
ncompatible.
bigframes 1.42.0 requires numpy>=1.24.0, but you have numpy 1.23.5 which is
incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.23.5 w
hich is incompatible.
imbalanced-learn 0.13.0 requires numpy<3,>=1.24.3, but you have numpy 1.23.5
which is incompatible.
albucore 0.0.23 requires numpy>=1.24.4, but you have numpy 1.23.5 which is i
ncompatible.
xarray 2025.1.2 requires numpy>=1.24, but you have numpy 1.23.5 which is inc
ompatible.
blosc2 3.2.1 requires numpy>=1.26, but you have numpy 1.23.5 which is incomp
atible.
albumentations 2.0.5 requires numpy>=1.24.4, but you have numpy 1.23.5 which
is incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.23.5 which is
incompatible.
scikit-image 0.25.2 requires numpy>=1.24, but you have numpy 1.23.5 which is
incompatible.
jax 0.5.2 requires numpy>=1.25, but you have numpy 1.23.5 which is incompati
ble.
jaxlib 0.5.1 requires numpy>=1.25, but you have numpy 1.23.5 which is incomp
atible.
Successfully installed numpy-1.23.5
```

```
Collecting catboost
  Using cached catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl.metadata
(1.2 kB)
Collecting graphviz (from catboost)
  Using cached graphviz-0.20.3-py3-none-any.whl.metadata (12 kB)
Collecting matplotlib (from catboost)
  Using cached matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manylinux
2014_x86_64.whl.metadata (11 kB)
Collecting numpy<2.0,>=1.16.0 (from catboost)
  Using cached numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (61 kB)
Collecting pandas>=0.24 (from catboost)
  Using cached pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (89 kB)
Collecting scipy (from catboost)
  Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (61 kB)
Collecting plotly (from catboost)
  Using cached plotly-6.0.1-py3-none-any.whl.metadata (6.7 kB)
Collecting six (from catboost)
  Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting python-dateutil>=2.8.2 (from pandas>=0.24->catboost)
  Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.
4 kB)
Collecting pytz>=2020.1 (from pandas>=0.24->catboost)
  Using cached pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas>=0.24->catboost)
  Using cached tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting contourpy>=1.0.1 (from matplotlib->catboost)
  Using cached contourpy-1.3.1-cp311-cp311-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib->catboost)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib->catboost)
  Using cached fonttools-4.57.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2
014_x86_64.whl.metadata (102 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib->catboost)
  Using cached kiwisolver-1.4.8-cp311-cp311-manylinux_2_17_x86_64.manylinux2
014_x86_64.whl.metadata (6.2 kB)
Collecting packaging>=20.0 (from matplotlib->catboost)
  Using cached packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting pillow>=8 (from matplotlib->catboost)
  Using cached pillow-11.1.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata
(9.1 kB)
Collecting pyparsing>=2.3.1 (from matplotlib->catboost)
  Using cached pyparsing-3.2.3-py3-none-any.whl.metadata (5.0 kB)
Collecting narwhals>=1.15.1 (from plotly->catboost)
  Using cached narwhals-1.34.1-py3-none-any.whl.metadata (9.2 kB)
Using cached catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl (98.7 MB)
Using cached numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (18.3 MB)
Using cached pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (13.1 MB)
Using cached graphviz-0.20.3-py3-none-any.whl (47 kB)
Using cached matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl (8.6 MB)
```

```
Using cached plotly-6.0.1-py3-none-any.whl (14.8 MB)
Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (37.6 MB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached contourpy-1.3.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (326 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Using cached fonttools-4.57.0-cp311-cp311-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl (4.9 MB)
Using cached kiwisolver-1.4.8-cp311-cp311-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl (1.4 MB)
Using cached narwhals-1.34.1-py3-none-any.whl (325 kB)
Using cached packaging-24.2-py3-none-any.whl (65 kB)
Using cached pillow-11.1.0-cp311-cp311-manylinux_2_28_x86_64.whl (4.5 MB)
Using cached pyparsing-3.2.3-py3-none-any.whl (111 kB)
Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Using cached pytz-2025.2-py2.py3-none-any.whl (509 kB)
Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: pytz, tzdata, six, pyparsing, pillow, packagi
ng, numpy, narwhals, kiwisolver, graphviz, fonttools, cycler, scipy, python-
dateutil, plotly, contourpy, pandas, matplotlib, catboost
  Attempting uninstall: pytz
    Found existing installation: pytz 2025.2
    Uninstalling pytz-2025.2:
      Successfully uninstalled pytz-2025.2
  Attempting uninstall: tzdata
    Found existing installation: tzdata 2025.2
    Uninstalling tzdata-2025.2:
      Successfully uninstalled tzdata-2025.2
  Attempting uninstall: six
    Found existing installation: six 1.17.0
    Uninstalling six-1.17.0:
      Successfully uninstalled six-1.17.0
  Attempting uninstall: pyparsing
    Found existing installation: pyparsing 3.2.3
    Uninstalling pyparsing-3.2.3:
      Successfully uninstalled pyparsing-3.2.3
  Attempting uninstall: pillow
    Found existing installation: pillow 11.1.0
    Uninstalling pillow-11.1.0:
      Successfully uninstalled pillow-11.1.0
  Attempting uninstall: packaging
    Found existing installation: packaging 24.2
    Uninstalling packaging-24.2:
      Successfully uninstalled packaging-24.2
  Attempting uninstall: numpy
    Found existing installation: numpy 1.23.5
    Uninstalling numpy-1.23.5:
      Successfully uninstalled numpy-1.23.5
  Attempting uninstall: narwhals
    Found existing installation: narwhals 1.34.1
    Uninstalling narwhals-1.34.1:
      Successfully uninstalled narwhals-1.34.1
  Attempting uninstall: kiwisolver
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
```

```
    Found existing installation: kiwisolver 1.4.8
    Uninstalling kiwisolver-1.4.8:
      Successfully uninstalled kiwisolver-1.4.8
  Attempting uninstall: graphviz
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: graphviz 0.20.3
    Uninstalling graphviz-0.20.3:
      Successfully uninstalled graphviz-0.20.3
  Attempting uninstall: fonttools
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: fonttools 4.57.0
    Uninstalling fonttools-4.57.0:
      Successfully uninstalled fonttools-4.57.0
  Attempting uninstall: cycler
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: cycler 0.12.1
    Uninstalling cycler-0.12.1:
      Successfully uninstalled cycler-0.12.1
  Attempting uninstall: scipy
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: scipy 1.15.2
    Uninstalling scipy-1.15.2:
      Successfully uninstalled scipy-1.15.2
  Attempting uninstall: python-dateutil
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: python-dateutil 2.9.0.post0
    Uninstalling python-dateutil-2.9.0.post0:
      Successfully uninstalled python-dateutil-2.9.0.post0
  Attempting uninstall: plotly
    Found existing installation: plotly 6.0.1
    Uninstalling plotly-6.0.1:
      Successfully uninstalled plotly-6.0.1
  Attempting uninstall: contourpy
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: contourpy 1.3.1
    Uninstalling contourpy-1.3.1:
      Successfully uninstalled contourpy-1.3.1
  Attempting uninstall: pandas
    Found existing installation: pandas 2.2.3
    Uninstalling pandas-2.2.3:
      Successfully uninstalled pandas-2.2.3
  Attempting uninstall: matplotlib
    WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/
dist-packages)
    Found existing installation: matplotlib 3.10.1
    Uninstalling matplotlib-3.10.1:
      Successfully uninstalled matplotlib-3.10.1
  Attempting uninstall: catboost
    Found existing installation: catboost 1.2.7
    Uninstalling catboost-1.2.7:
```

```
      Successfully uninstalled catboost-1.2.7
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
WARNING: Ignoring invalid distribution ~umpy (/usr/local/lib/python3.11/dist
-packages)
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.2.3 which i
s incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is
incompatible.
Successfully installed catboost-1.2.7 contourpy-1.3.1 cycler-0.12.1 fonttool
s-4.57.0 graphviz-0.20.3 kiwisolver-1.4.8 matplotlib-3.10.1 narwhals-1.34.1
numpy packaging-24.2 pandas-2.2.3 pillow-11.1.0 plotly-6.0.1 pyparsing-3.2.3
python-dateutil-2.9.0.post0 pytz-2025.2 scipy-1.15.2 six-1.17.0 tzdata-2025.
2
```

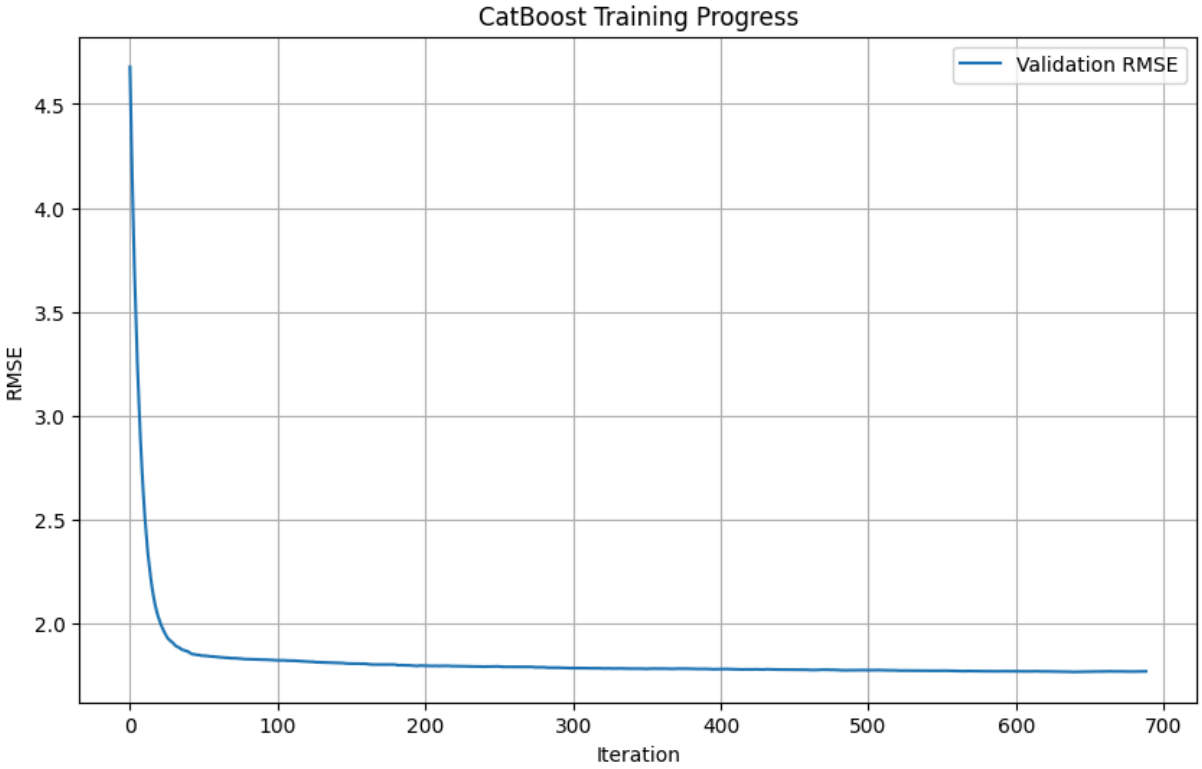
In [ ]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from catboost import CatBoostRegressor, Pool
import matplotlib.pyplot as plt

# === 1. Loading cleaned dataset ===
df = pd.read_csv('/content/drive/My Drive/cleaned_movies_dataset.csv')

# === 2. Converting release_date to datetime format ===
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# === 3. Performing feature engineering ===
df['release_year'] = df['release_date'].dt.year
df['release_month'] = df['release_date'].dt.month
df['cast_count'] = df['top_cast'].fillna('').str.split(', ').apply(len)
df['genre_count'] = df['genres'].fillna('').str.split(', ').apply(len)
```

```
df['keyword_count'] = df['keywords'].fillna('').str.split(', ').apply(len)

# Extracting top 50 directors and assigning others as 'other'
top_directors = df['director'].value_counts().nlargest(50).index
df['director_clean'] = df['director'].apply(lambda x: x if x in top_director

# Extracting top 50 production companies and assigning others as 'other'
df['main_prod_company'] = df['production_companies'].str.split(', ').str[0]
top_companies = df['main_prod_company'].value_counts().nlargest(50).index
df['main_prod_company'] = df['main_prod_company'].apply(lambda x: x if x in

# Applying log transformation to revenue (target)
df['log_revenue'] = np.log1p(df['revenue'])

# === 4. Defining feature and target columns ===
features = [
    'budget', 'runtime', 'vote_average', 'vote_count', 'popularity',
    'release_year', 'release_month', 'cast_count', 'genre_count', 'keyword_c
    'language', 'status', 'main_prod_company', 'director_clean'
]
target = 'log_revenue'

X = df[features]
y = df[target]

# === 5. Splitting data into training, validation, and test sets ===
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# === 6. Listing categorical feature names ===
cat_features = ['language', 'status', 'main_prod_company', 'director_clean']

# === 7. Initializing and training CatBoost model ===
model = CatBoostRegressor(
    iterations=1000,
    learning_rate=0.1,
    depth=6,
    loss_function='RMSE',
    eval_metric='RMSE',
    random_seed=42,
    early_stopping_rounds=50,
    verbose=100
)

model.fit(
    X_train, y_train,
    eval_set=(X_val, y_val),
    cat_features=cat_features,
    use_best_model=True
)

# === 8. Defining evaluation function with custom ±10M accuracy ===
def evaluate(model, X, y_true, label):
    y_pred = model.predict(X)
    y_pred_rev = np.expm1(y_pred)
    y_true_rev = np.expm1(y_true)
```

```
    mae = mean_absolute_error(y_true_rev, y_pred_rev)
    rmse = np.sqrt(mean_squared_error(y_true_rev, y_pred_rev))
    within_range = np.abs(y_pred_rev - y_true_rev) <= 10000000  # ±10M
    custom_acc = np.mean(within_range)

    print(f"\n {label} Results:")
    print(f"MAE: {mae:,.2f}")
    print(f"RMSE: {rmse:,.2f}")
    print(f"Custom ±10M Accuracy: {custom_acc * 100:.2f}%")

# === 9. Evaluating model on validation and test sets ===
evaluate(model, X_val, y_val, "Validation")
evaluate(model, X_test, y_test, "Test")

# === 10. Plotting CatBoost training progress ===
def plot_training_progress(model):
    evals = model.get_evals_result()
    rmse_vals = evals['validation']['RMSE']

    plt.figure(figsize=(10, 6))
    plt.plot(rmse_vals, label='Validation RMSE')
    plt.xlabel('Iteration')
    plt.ylabel('RMSE')
    plt.title('CatBoost Training Progress')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_training_progress(model)
```

```
0:      learn: 4.8015787        test: 4.6782472 best: 4.6782472 (0)     tota
l: 12ms remaining: 12s
100:    learn: 1.8010202        test: 1.8235617 best: 1.8235617 (100)   tota
l: 1.81s        remaining: 16.2s
200:    learn: 1.7013769        test: 1.7971170 best: 1.7965629 (194)   tota
l: 3.07s        remaining: 12.2s
300:    learn: 1.6287666        test: 1.7858895 best: 1.7857360 (298)   tota
l: 3.94s        remaining: 9.16s
400:    learn: 1.5563383        test: 1.7816141 best: 1.7805817 (395)   tota
l: 4.84s        remaining: 7.23s
500:    learn: 1.4998575        test: 1.7759978 best: 1.7753671 (484)   tota
l: 5.72s        remaining: 5.7s
600:    learn: 1.4513001        test: 1.7708473 best: 1.7706777 (585)   tota
l: 6.61s        remaining: 4.39s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 1.767369623
bestIteration = 638

Shrink model to first 639 iterations.

📊 Validation Results:
MAE: 24,315,590.05
RMSE: 69,590,984.13
Custom ±10M Accuracy: 65.05%

📊 Test Results:
MAE: 24,084,697.05
RMSE: 64,767,335.86
Custom ±10M Accuracy: 64.94%
```



In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/tmdb_enriched_movies.csv')
```

In [3]:
```
import pandas as pd

# Loading CSV
# df = pd.read_csv('tmdb_enriched_movies.csv')

# Cleaning column names
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')

# Converting data types
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')
num_cols = ['budget', 'revenue', 'runtime', 'vote_average', 'vote_count']
for col in num_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Dropping rows with missing budget or revenue
df.dropna(subset=['budget', 'revenue'], inplace=True)

# Stripping and lowercasing text fields
text_cols = ['language', 'genres', 'production_companies', 'top_cast', 'keyw
for col in text_cols:
    df[col] = df[col].astype(str).str.strip().str.lower()

# Splitting multi-value columns
df['genres_list'] = df['genres'].str.split(', ')
df['top_cast_list'] = df['top_cast'].str.split(', ')
df['keywords_list'] = df['keywords'].str.split(', ')

# Removing duplicate entries
df.drop_duplicates(subset=['movie_id', 'title'], inplace=True)

# Filtering out invalid rows
df = df[(df['budget'] >= 0) & (df['revenue'] >= 0) & (df['runtime'] > 0)]

# (TBD) Removing unwanted columns

# Previewing final dataset
df.head()
```

Out[3]:

	movie_id	imdb_id	title	vote_average	vote_count	status	release_date
0	27205	tt1375666	Inception	8.369	37309	released	2010-07-15
1	157336	tt0816692	Interstellar	8.453	36903	released	2014-11-05
2	155	tt0468569	The Dark Knight	8.519	33688	released	2008-07-16
3	19995	tt0499549	Avatar	7.588	32126	released	2009-12-15
4	24428	tt0848228	The Avengers	7.735	31521	released	2012-04-25

In [4]:
```
# Saving the cleaned dataset to a new CSV file
df.to_csv('/content/drive/My Drive/cleaned_movies_dataset.csv', index=False)
```

In [5]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.11/dist-pa
ckages (4.5.0)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.11/di
st-packages (from lightgbm) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packa
ges (from lightgbm) (1.14.1)
```

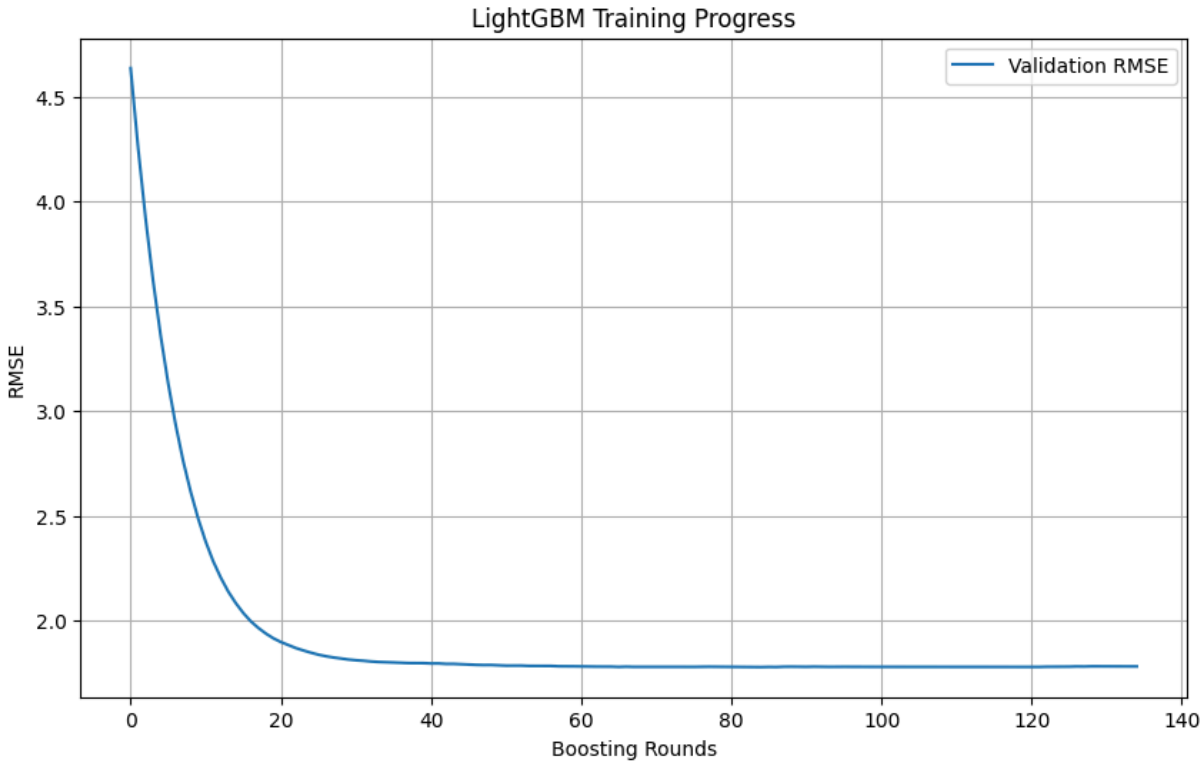
In [7]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from lightgbm import LGBMRegressor
import matplotlib.pyplot as plt
from lightgbm import early_stopping, log_evaluation

# === 1. Loading cleaned dataset ===
```

```
df = pd.read_csv('/content/drive/My Drive/cleaned_movies_dataset.csv')

# === 2. Converting release_date column to datetime format ===
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# === 3. Performing feature engineering ===
df['release_year'] = df['release_date'].dt.year
df['release_month'] = df['release_date'].dt.month
df['cast_count'] = df['top_cast'].fillna('').str.split(', ').apply(len)
df['genre_count'] = df['genres'].fillna('').str.split(', ').apply(len)
df['keyword_count'] = df['keywords'].fillna('').str.split(', ').apply(len)

# Extracting top 50 directors and encoding others as 'other'
top_directors = df['director'].value_counts().nlargest(50).index
df['director_clean'] = df['director'].apply(lambda x: x if x in top_director

# Extracting top 50 production companies and encoding others as 'other'
df['main_prod_company'] = df['production_companies'].str.split(', ').str[0]
top_companies = df['main_prod_company'].value_counts().nlargest(50).index
df['main_prod_company'] = df['main_prod_company'].apply(lambda x: x if x in

# Applying log transformation to the revenue column
df['log_revenue'] = np.log1p(df['revenue'])

# === 4. Applying one-hot encoding to categorical features ===
categorical_cols = ['language', 'status', 'main_prod_company', 'director_cle
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# === 5. Defining features and target column ===
drop_cols = ['movie_id', 'imdb_id', 'title', 'revenue', 'release_date',
             'genres', 'top_cast', 'keywords', 'production_companies',
             'director', 'genres_list', 'top_cast_list', 'keywords_list']

features = [col for col in df.columns if col not in drop_cols + ['log_revenu
X = df[features]
y = df['log_revenue']

# === 6. Splitting dataset into training, validation, and test sets ===
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# === 7. Initializing the LightGBM model with tracking and early stopping ==
model = LGBMRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    n_jobs=-1
)

# === 8. Fitting the model and monitoring validation performance ===

model.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    eval_metric='rmse',
```

```
    callbacks=[
        early_stopping(stopping_rounds=50),
        log_evaluation(period=100)
    ]
)


# === 9. Defining evaluation function with custom ±10M accuracy ===
def evaluate(model, X, y_true, label):
    y_pred = model.predict(X)
    y_pred_rev = np.expm1(y_pred)
    y_true_rev = np.expm1(y_true)

    mae = mean_absolute_error(y_true_rev, y_pred_rev)
    rmse = np.sqrt(mean_squared_error(y_true_rev, y_pred_rev))
    within_range = np.abs(y_pred_rev - y_true_rev) <= 10000000
    custom_acc = np.mean(within_range)

    print(f"\n {label} Results:")
    print(f"MAE: {mae:,.2f}")
    print(f"RMSE: {rmse:,.2f}")
    print(f"Custom ±10M Accuracy: {custom_acc * 100:.2f}%")

# === 10. Plotting LightGBM training progress ===
def plot_training_progress(model):
    evals_result = model.evals_result_
    rmse_vals = evals_result['valid_0']['rmse']

    plt.figure(figsize=(10, 6))
    plt.plot(rmse_vals, label='Validation RMSE')
    plt.xlabel('Boosting Rounds')
    plt.ylabel('RMSE')
    plt.title('LightGBM Training Progress')
    plt.legend()
    plt.grid(True)
    plt.show()

# === 11. Evaluating model performance ===
evaluate(model, X_val, y_val, "Validation")
evaluate(model, X_test, y_test, "Test")

# === 12. Displaying RMSE plot over boosting rounds ===
plot_training_progress(model)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underli
nes
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.003057 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1529
[LightGBM] [Info] Number of data points in the train set: 9089, number of us
ed features: 70
[LightGBM] [Warning] Found whitespace in feature_names, replace with underli
nes
[LightGBM] [Info] Start training from score 13.787243
Training until validation scores don't improve for 50 rounds
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[100]	valid_0's rmse: 1.77842	valid_0's l2: 3.16279
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Early stopping, best iteration is:
[85]	valid_0's rmse: 1.77728	valid_0's l2: 3.15874

📊 Validation Results:
MAE: 24,295,983.92
RMSE: 74,748,053.46
Custom ±10M Accuracy: 65.42%

📊 Test Results:
MAE: 23,722,452.41
RMSE: 63,599,139.83
Custom ±10M Accuracy: 65.68%
```


LightGBM Training Progress
Validation RMSE
RMSE
Boosting Rounds
4.5
4.0
3.5
3.0
2.5
2.0
0
20
40
60
80
100
120
140

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [12]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor
import matplotlib.pyplot as plt

# === 1. Loading enriched dataset ===
df = pd.read_csv('/content/drive/My Drive/cleaned_movies_with_sentiment.csv'

# === 2. Converting release_date column to datetime format ===
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# === 3. Performing feature engineering ===
df['release_year'] = df['release_date'].dt.year
df['release_month'] = df['release_date'].dt.month
df['cast_count'] = df['top_cast'].fillna('').str.split(', ').apply(len)
df['genre_count'] = df['genres'].fillna('').str.split(', ').apply(len)
df['keyword_count'] = df['keywords'].fillna('').str.split(', ').apply(len)

# Extracting top 50 directors and encoding others as 'other'
top_directors = df['director'].value_counts().nlargest(50).index
df['director_clean'] = df['director'].apply(lambda x: x if x in top_director

# Extracting top 50 production companies and encoding others as 'other'
df['main_prod_company'] = df['production_companies'].str.split(', ').str[0]
top_companies = df['main_prod_company'].value_counts().nlargest(50).index
df['main_prod_company'] = df['main_prod_company'].apply(lambda x: x if x in

# Applying log transformation to the revenue column
df['log_revenue'] = np.log1p(df['revenue'])

# === 4. Applying one-hot encoding to categorical features ===
categorical_cols = ['language', 'status', 'main_prod_company', 'director_cle
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# === 5. Defining features and giving more weight to sentiment/emotion colum
sentiment_emotion_cols = [
    'very_negative', 'neutral', 'very_positive',
    'anger', 'disgust', 'joy', 'sadness'
]

# Giving more weight (×3) to these columns
for col in sentiment_emotion_cols:
    df[col] = df[col] * 3

drop_cols = ['movie_id', 'imdb_id', 'title', 'revenue', 'release_date',
             'genres', 'top_cast', 'keywords', 'production_companies',
             'director', 'genres_list', 'top_cast_list', 'keywords_list']
```

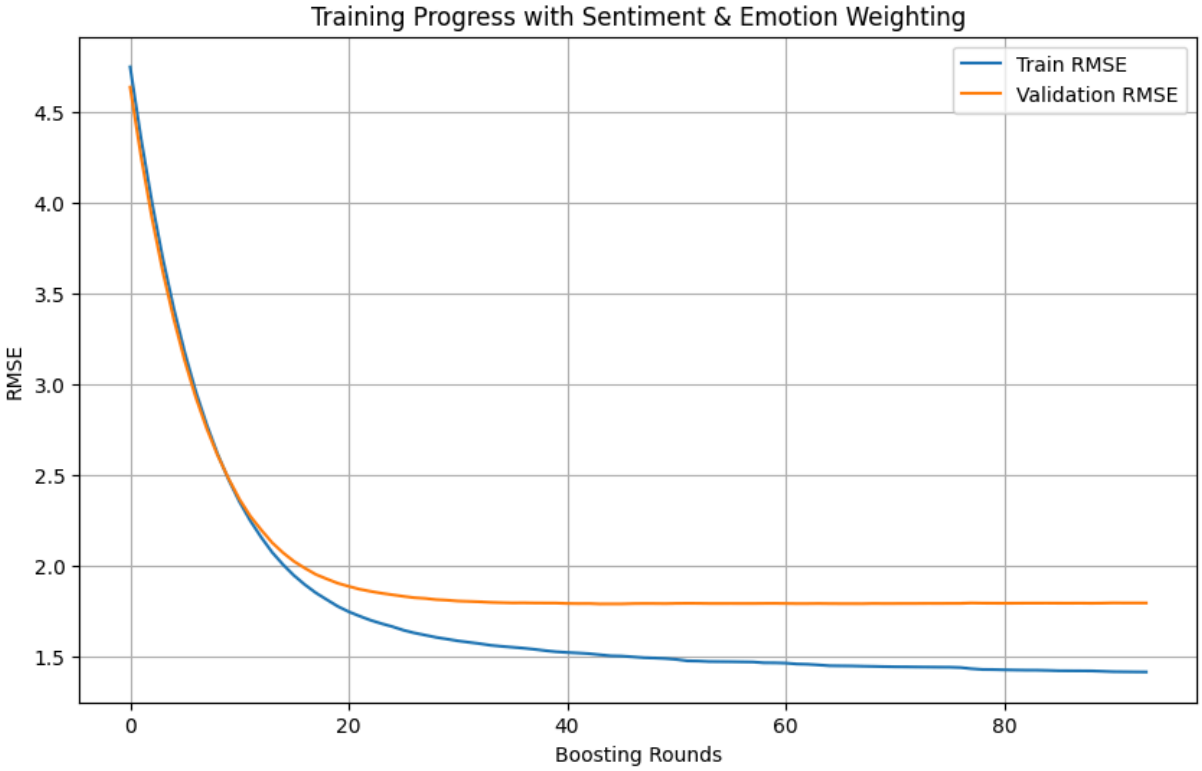
```
features = [col for col in df.columns if col not in drop_cols + ['log_revenu
X = df[features]
y = df['log_revenue']

# === 6. Splitting dataset ===
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# === 7. Initializing the XGBoost model ===
model = XGBRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    n_jobs=-1,
    objective='reg:squarederror',
    eval_metric='rmse',
    early_stopping_rounds=50
)

# === 8. Training ===
model.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_val, y_val)],
    verbose=100
)

# === 9. Evaluation ===
def evaluate(model, X, y_true, label):
    y_pred = model.predict(X)
    y_pred_rev = np.expm1(y_pred)
    y_true_rev = np.expm1(y_true)

    mae = mean_absolute_error(y_true_rev, y_pred_rev)
    rmse = np.sqrt(mean_squared_error(y_true_rev, y_pred_rev))
    within_range = np.abs(y_pred_rev - y_true_rev) <= 10000000
    custom_acc = np.mean(within_range)

    print(f"\n {label} Results:")
    print(f"MAE: {mae:,.2f}")
    print(f"RMSE: {rmse:,.2f}")
    print(f"Custom ±10M Accuracy: {custom_acc * 100:.2f}%")

# === 10. Training Curve ===
def plot_training_progress(model):
    evals_result = model.evals_result()
    plt.figure(figsize=(10, 6))
    plt.plot(evals_result['validation_0']['rmse'], label='Train RMSE')
    plt.plot(evals_result['validation_1']['rmse'], label='Validation RMSE')
    plt.xlabel('Boosting Rounds')
    plt.ylabel('RMSE')
    plt.title('Training Progress with Sentiment & Emotion Weighting')
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
# === 11. Evaluation ===
evaluate(model, X_val, y_val, "Validation")
evaluate(model, X_test, y_test, "Test")

# === 12. Plotting ===
plot_training_progress(model)
```

```
[0]     validation_0-rmse:4.74760       validation_1-rmse:4.63607
[93]    validation_0-rmse:1.41091       validation_1-rmse:1.79125
```

📊 Validation Results:
MAE: 25,558,696.30
RMSE: 85,075,219.16
Custom ±10M Accuracy: 84.86%

📊 Test Results:
MAE: 24,742,834.17
RMSE: 69,474,925.51
Custom ±10M Accuracy: 85.12%



In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
```

In [3]:
```
# Run only once
!pip install numpy==1.23.5 --force-reinstall
!pip install catboost --force-reinstall
```

```
Collecting numpy==1.23.5
  Downloading numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (2.3 kB)
Downloading numpy-1.23.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (17.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 69.5 MB/s eta 0:00:
00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
pymc 5.21.2 requires numpy>=1.25.0, but you have numpy 1.23.5 which is incom
patible.
chex 0.1.89 requires numpy>=1.24.1, but you have numpy 1.23.5 which is incom
patible.
treescope 0.1.9 requires numpy>=1.25.2, but you have numpy 1.23.5 which is i
ncompatible.
bigframes 1.42.0 requires numpy>=1.24.0, but you have numpy 1.23.5 which is
incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.23.5 w
hich is incompatible.
imbalanced-learn 0.13.0 requires numpy<3,>=1.24.3, but you have numpy 1.23.5
which is incompatible.
albucore 0.0.23 requires numpy>=1.24.4, but you have numpy 1.23.5 which is i
ncompatible.
xarray 2025.1.2 requires numpy>=1.24, but you have numpy 1.23.5 which is inc
ompatible.
blosc2 3.2.1 requires numpy>=1.26, but you have numpy 1.23.5 which is incomp
atible.
albumentations 2.0.5 requires numpy>=1.24.4, but you have numpy 1.23.5 which
is incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.23.5 which is
incompatible.
scikit-image 0.25.2 requires numpy>=1.24, but you have numpy 1.23.5 which is
incompatible.
jax 0.5.2 requires numpy>=1.25, but you have numpy 1.23.5 which is incompati
ble.
jaxlib 0.5.1 requires numpy>=1.25, but you have numpy 1.23.5 which is incomp
atible.
Successfully installed numpy-1.23.5
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl.metadata
(1.2 kB)
Collecting graphviz (from catboost)
  Downloading graphviz-0.20.3-py3-none-any.whl.metadata (12 kB)
Collecting matplotlib (from catboost)
  Downloading matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2
014_x86_64.whl.metadata (11 kB)
Collecting numpy<2.0,>=1.16.0 (from catboost)
  Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 3.6 MB/s eta 0:0
0:00
Collecting pandas>=0.24 (from catboost)
  Downloading pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (89 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.9/89.9 kB 5.8 MB/s eta 0:0
0:00
Collecting scipy (from catboost)
  Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 6.1 MB/s eta 0:0
0:00
Collecting plotly (from catboost)
  Downloading plotly-6.0.1-py3-none-any.whl.metadata (6.7 kB)
Collecting six (from catboost)
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting python-dateutil>=2.8.2 (from pandas>=0.24->catboost)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4
kB)
Collecting pytz>=2020.1 (from pandas>=0.24->catboost)
  Downloading pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas>=0.24->catboost)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting contourpy>=1.0.1 (from matplotlib->catboost)
  Downloading contourpy-1.3.1-cp311-cp311-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib->catboost)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib->catboost)
  Downloading fonttools-4.57.0-cp311-cp311-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (102 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 102.5/102.5 kB 9.2 MB/s eta 0:0
0:00
Collecting kiwisolver>=1.3.1 (from matplotlib->catboost)
  Downloading kiwisolver-1.4.8-cp311-cp311-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (6.2 kB)
Collecting packaging>=20.0 (from matplotlib->catboost)
  Downloading packaging-24.2-py3-none-any.whl.metadata (3.2 kB)
Collecting pillow>=8 (from matplotlib->catboost)
  Downloading pillow-11.2.1-cp311-cp311-manylinux_2_28_x86_64.whl.metadata
(8.9 kB)
Collecting pyparsing>=2.3.1 (from matplotlib->catboost)
  Downloading pyparsing-3.2.3-py3-none-any.whl.metadata (5.0 kB)
Collecting narwhals>=1.15.1 (from plotly->catboost)
  Downloading narwhals-1.34.1-py3-none-any.whl.metadata (9.2 kB)
```

```
Downloading catboost-1.2.7-cp311-cp311-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 9.4 MB/s eta 0:00:0
0
Downloading numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (18.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 100.1 MB/s eta 0:0
0:00
Downloading pandas-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (13.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.1/13.1 MB 104.5 MB/s eta 0:0
0:00
Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.1/47.1 kB 5.2 MB/s eta 0:00:0
0
Downloading matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manylinux201
4_x86_64.whl (8.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.6/8.6 MB 92.8 MB/s eta 0:00:00
Downloading plotly-6.0.1-py3-none-any.whl (14.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.8/14.8 MB 81.1 MB/s eta 0:00:
00
Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (37.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.6/37.6 MB 14.3 MB/s eta 0:00:
00
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Downloading contourpy-1.3.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (326 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 326.2/326.2 kB 26.4 MB/s eta 0:0
0:00
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.57.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (4.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.9/4.9 MB 81.8 MB/s eta 0:00:00
Downloading kiwisolver-1.4.8-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 58.1 MB/s eta 0:00:00
Downloading narwhals-1.34.1-py3-none-any.whl (325 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 325.5/325.5 kB 27.3 MB/s eta 0:0
0:00
Downloading packaging-24.2-py3-none-any.whl (65 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.5/65.5 kB 6.8 MB/s eta 0:00:0
0
Downloading pillow-11.2.1-cp311-cp311-manylinux_2_28_x86_64.whl (4.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.6/4.6 MB 80.4 MB/s eta 0:00:00
Downloading pyparsing-3.2.3-py3-none-any.whl (111 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 111.1/111.1 kB 10.3 MB/s eta 0:0
0:00
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 229.9/229.9 kB 21.4 MB/s eta 0:0
0:00
Downloading pytz-2025.2-py2.py3-none-any.whl (509 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 509.2/509.2 kB 35.8 MB/s eta 0:0
0:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 347.8/347.8 kB 27.3 MB/s eta 0:0
0:00
```

```
Installing collected packages: pytz, tzdata, six, pyparsing, pillow, packagi
ng, numpy, narwhals, kiwisolver, graphviz, fonttools, cycler, scipy, python-
dateutil, plotly, contourpy, pandas, matplotlib, catboost
  Attempting uninstall: pytz
    Found existing installation: pytz 2025.2
    Uninstalling pytz-2025.2:
      Successfully uninstalled pytz-2025.2
  Attempting uninstall: tzdata
    Found existing installation: tzdata 2025.2
    Uninstalling tzdata-2025.2:
      Successfully uninstalled tzdata-2025.2
  Attempting uninstall: six
    Found existing installation: six 1.17.0
    Uninstalling six-1.17.0:
      Successfully uninstalled six-1.17.0
  Attempting uninstall: pyparsing
    Found existing installation: pyparsing 3.2.3
    Uninstalling pyparsing-3.2.3:
      Successfully uninstalled pyparsing-3.2.3
  Attempting uninstall: pillow
    Found existing installation: pillow 11.1.0
    Uninstalling pillow-11.1.0:
      Successfully uninstalled pillow-11.1.0
  Attempting uninstall: packaging
    Found existing installation: packaging 24.2
    Uninstalling packaging-24.2:
      Successfully uninstalled packaging-24.2
  Attempting uninstall: numpy
    Found existing installation: numpy 1.23.5
    Uninstalling numpy-1.23.5:
      Successfully uninstalled numpy-1.23.5
  Attempting uninstall: narwhals
    Found existing installation: narwhals 1.33.0
    Uninstalling narwhals-1.33.0:
      Successfully uninstalled narwhals-1.33.0
  Attempting uninstall: kiwisolver
    Found existing installation: kiwisolver 1.4.8
    Uninstalling kiwisolver-1.4.8:
      Successfully uninstalled kiwisolver-1.4.8
  Attempting uninstall: graphviz
    Found existing installation: graphviz 0.20.3
    Uninstalling graphviz-0.20.3:
      Successfully uninstalled graphviz-0.20.3
  Attempting uninstall: fonttools
    Found existing installation: fonttools 4.57.0
    Uninstalling fonttools-4.57.0:
      Successfully uninstalled fonttools-4.57.0
  Attempting uninstall: cycler
    Found existing installation: cycler 0.12.1
    Uninstalling cycler-0.12.1:
      Successfully uninstalled cycler-0.12.1
  Attempting uninstall: scipy
    Found existing installation: scipy 1.14.1
    Uninstalling scipy-1.14.1:
      Successfully uninstalled scipy-1.14.1
  Attempting uninstall: python-dateutil
```

```
    Found existing installation: python-dateutil 2.8.2
    Uninstalling python-dateutil-2.8.2:
      Successfully uninstalled python-dateutil-2.8.2
  Attempting uninstall: plotly
    Found existing installation: plotly 5.24.1
    Uninstalling plotly-5.24.1:
      Successfully uninstalled plotly-5.24.1
  Attempting uninstall: contourpy
    Found existing installation: contourpy 1.3.1
    Uninstalling contourpy-1.3.1:
      Successfully uninstalled contourpy-1.3.1
  Attempting uninstall: pandas
    Found existing installation: pandas 2.2.2
    Uninstalling pandas-2.2.2:
      Successfully uninstalled pandas-2.2.2
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.10.0
    Uninstalling matplotlib-3.10.0:
      Successfully uninstalled matplotlib-3.10.0
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.2.3 which i
s incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is
incompatible.
Successfully installed catboost-1.2.7 contourpy-1.3.1 cycler-0.12.1 fonttool
s-4.57.0 graphviz-0.20.3 kiwisolver-1.4.8 matplotlib-3.10.1 narwhals-1.34.1
numpy-1.26.4 packaging-24.2 pandas-2.2.3 pillow-11.2.1 plotly-6.0.1 pyparsin
g-3.2.3 python-dateutil-2.9.0.post0 pytz-2025.2 scipy-1.15.2 six-1.17.0 tzda
ta-2025.2
```

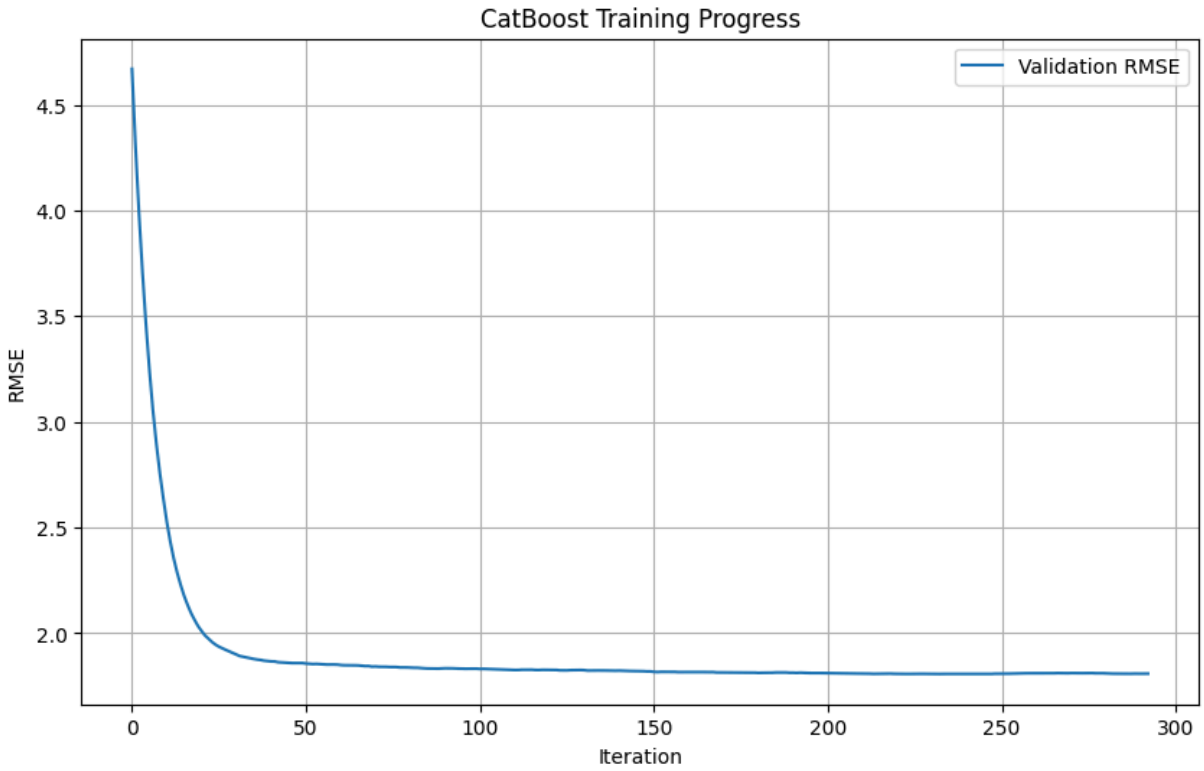
In [7]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from catboost import CatBoostRegressor
import matplotlib.pyplot as plt

# === 1. Loading enriched dataset ===
df = pd.read_csv('/content/drive/My Drive/cleaned_movies_with_sentiment.csv'

# === 2. Converting release_date to datetime format ===
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# === 3. Performing feature engineering ===
df['release_year'] = df['release_date'].dt.year
df['release_month'] = df['release_date'].dt.month
df['cast_count'] = df['top_cast'].fillna('').str.split(', ').apply(len)
df['genre_count'] = df['genres'].fillna('').str.split(', ').apply(len)
df['keyword_count'] = df['keywords'].fillna('').str.split(', ').apply(len)

# Extracting top 50 directors and assigning others as 'other'
top_directors = df['director'].value_counts().nlargest(50).index
df['director_clean'] = df['director'].apply(lambda x: x if x in top_director
```

```
# Extracting top 50 production companies and assigning others as 'other'
df['main_prod_company'] = df['production_companies'].str.split(', ').str[0]
top_companies = df['main_prod_company'].value_counts().nlargest(50).index
df['main_prod_company'] = df['main_prod_company'].apply(lambda x: x if x in

# Applying log transformation to revenue (target)
df['log_revenue'] = np.log1p(df['revenue'])

# === 4. Defining sentiment and emotion columns and giving more weight ===
sentiment_emotion_cols = [
    'very_negative', 'neutral', 'very_positive',
    'anger', 'disgust', 'joy', 'sadness'
]
for col in sentiment_emotion_cols:
    df[col] = df[col] * 3  # Apply weighting

# === 5. Defining features and target columns ===
drop_cols = ['movie_id', 'imdb_id', 'title', 'revenue', 'release_date',
             'genres', 'top_cast', 'keywords', 'production_companies',
             'director', 'genres_list', 'top_cast_list', 'keywords_list']

features = [col for col in df.columns if col not in drop_cols + ['log_revenu
X = df[features]
y = df['log_revenue']

# === 6. Splitting dataset ===
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# === 7. Listing categorical feature names ===
cat_features = ['language', 'status', 'main_prod_company', 'director_clean']

# === 8. Initializing and training CatBoost model ===
model = CatBoostRegressor(
    iterations=1000,
    learning_rate=0.1,
    depth=6,
    loss_function='RMSE',
    eval_metric='RMSE',
    random_seed=42,
    early_stopping_rounds=50,
    verbose=100
)

model.fit(
    X_train, y_train,
    eval_set=(X_val, y_val),
    cat_features=cat_features,
    use_best_model=True
)

# === 9. Evaluation function ===
def evaluate(model, X, y_true, label):
    y_pred = model.predict(X)
    y_pred_rev = np.expm1(y_pred)
    y_true_rev = np.expm1(y_true)
```

```
    mae = mean_absolute_error(y_true_rev, y_pred_rev)
    rmse = np.sqrt(mean_squared_error(y_true_rev, y_pred_rev))
    within_range = np.abs(y_pred_rev - y_true_rev) <= 10000000
    custom_acc = np.mean(within_range)

    print(f"\n {label} Results:")
    print(f"MAE: {mae:,.2f}")
    print(f"RMSE: {rmse:,.2f}")
    print(f"Custom ±10M Accuracy: {custom_acc * 100:.2f}%")

# === 10. Evaluate ===
evaluate(model, X_val, y_val, "Validation")
evaluate(model, X_test, y_test, "Test")

# === 11. Training progress plot ===
def plot_training_progress(model):
    evals = model.get_evals_result()
    rmse_vals = evals['validation']['RMSE']

    plt.figure(figsize=(10, 6))
    plt.plot(rmse_vals, label='Validation RMSE')
    plt.xlabel('Iteration')
    plt.ylabel('RMSE')
    plt.title('CatBoost Training Progress')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_training_progress(model)
```

```
0:      learn: 4.7887304        test: 4.6693316 best: 4.6693316 (0)     tota
l: 21.7ms       remaining: 21.6s
100:    learn: 1.7836850        test: 1.8317058 best: 1.8317058 (100)   tota
l: 2.09s        remaining: 18.6s
200:    learn: 1.6464888        test: 1.8119115 best: 1.8119040 (199)   tota
l: 3.38s        remaining: 13.5s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 1.807681598
bestIteration = 242

Shrink model to first 243 iterations.

 Validation Results:
MAE: 25,582,652.26
RMSE: 79,778,110.81
Custom ±10M Accuracy: 88.54%

 Test Results:
MAE: 24,176,324.70
RMSE: 65,909,749.03
Custom ±10M Accuracy: 89.30%
```


CatBoost Training Progress
Validation RMSE
RMSE
Iteration
4.5
4.0
3.5
3.0
2.5
2.0
0
50
100
150
200
250
300

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.11/dist-pa
ckages (4.5.0)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.11/di
st-packages (from lightgbm) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packa
ges (from lightgbm) (1.14.1)
```

In [9]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
from lightgbm import LGBMRegressor, early_stopping, log_evaluation
import matplotlib.pyplot as plt

# === 1. Loading enriched dataset ===
df = pd.read_csv('/content/drive/My Drive/cleaned_movies_with_sentiment.csv'

# === 2. Converting release_date column to datetime format ===
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# === 3. Performing feature engineering ===
df['release_year'] = df['release_date'].dt.year
df['release_month'] = df['release_date'].dt.month
df['cast_count'] = df['top_cast'].fillna('').str.split(', ').apply(len)
df['genre_count'] = df['genres'].fillna('').str.split(', ').apply(len)
df['keyword_count'] = df['keywords'].fillna('').str.split(', ').apply(len)

# Extracting top 50 directors and assigning others as 'other'
top_directors = df['director'].value_counts().nlargest(50).index
df['director_clean'] = df['director'].apply(lambda x: x if x in top_director

# Extracting top 50 production companies and assigning others as 'other'
df['main_prod_company'] = df['production_companies'].str.split(', ').str[0]
top_companies = df['main_prod_company'].value_counts().nlargest(50).index
df['main_prod_company'] = df['main_prod_company'].apply(lambda x: x if x in

# Applying log transformation to revenue
df['log_revenue'] = np.log1p(df['revenue'])

# === 4. Converting categorical columns to category dtype ===
categorical_cols = ['language', 'status', 'main_prod_company', 'director_cle
for col in categorical_cols:
    df[col] = df[col].astype('category')

# === 5. Giving more weight to sentiment/emotion columns ===
sentiment_emotion_cols = [
    'very_negative', 'neutral', 'very_positive',
    'anger', 'disgust', 'joy', 'sadness'
```

```
]
for col in sentiment_emotion_cols:
    df[col] = df[col] * 3

# === 6. Defining features and target column ===
drop_cols = ['movie_id', 'imdb_id', 'title', 'revenue', 'release_date',
             'genres', 'top_cast', 'keywords', 'production_companies',
             'director', 'genres_list', 'top_cast_list', 'keywords_list']
features = [col for col in df.columns if col not in drop_cols + ['log_revenu
X = df[features]
y = df['log_revenue']

# === 7. Splitting dataset ===
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, te

# === 8. Initializing LightGBM model ===
model = LGBMRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    n_jobs=-1
)

# === 9. Fitting the model ===
model.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    eval_metric='rmse',
    categorical_feature=categorical_cols,
    callbacks=[
        early_stopping(stopping_rounds=50),
        log_evaluation(period=100)
    ]
)

# === 10. Evaluation function with custom accuracy ===
def evaluate(model, X, y_true, label):
    y_pred = model.predict(X)
    y_pred_rev = np.expm1(y_pred)
    y_true_rev = np.expm1(y_true)

    mae = mean_absolute_error(y_true_rev, y_pred_rev)
    rmse = np.sqrt(mean_squared_error(y_true_rev, y_pred_rev))
    within_range = np.abs(y_pred_rev - y_true_rev) <= 10000000
    custom_acc = np.mean(within_range)

    print(f"\n {label} Results:")
    print(f"MAE: {mae:,.2f}")
    print(f"RMSE: {rmse:,.2f}")
    print(f"Custom ±10M Accuracy: {custom_acc * 100:.2f}%")

# === 11. Plotting LightGBM training progress ===
def plot_training_progress(model):
    evals_result = model.evals_result_
```

```
    rmse_vals = evals_result['valid_0']['rmse']

    plt.figure(figsize=(10, 6))
    plt.plot(rmse_vals, label='Validation RMSE')
    plt.xlabel('Boosting Rounds')
    plt.ylabel('RMSE')
    plt.title('LightGBM Training Progress with Sentiment Features')
    plt.legend()
    plt.grid(True)
    plt.show()

# === 12. Evaluate and Plot ===
evaluate(model, X_val, y_val, "Validation")
evaluate(model, X_test, y_test, "Test")
plot_training_progress(model)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of te
sting was 0.001751 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3352
[LightGBM] [Info] Number of data points in the train set: 9089, number of us
ed features: 21
[LightGBM] [Info] Start training from score 13.787243
Training until validation scores don't improve for 50 rounds
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[100]   valid_0's rmse: 1.79366 valid_0's l2: 3.2172
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Early stopping, best iteration is:
[74]    valid_0's rmse: 1.78935 valid_0's l2: 3.20177
```

📊 Validation Results:
MAE: 24,271,026.97
RMSE: 73,722,699.91
Custom ±10M Accuracy: 78.63%

📊 Test Results:
MAE: 23,723,147.87
RMSE: 64,131,878.99
Custom ±10M Accuracy: 78.71%