Project of Data Analytics Internship

On

“Movie Success Prediction and Sentiment Study”

ELEVATE LABS

18th May 2025

ACKNOWLEDGEMENT

The Data Analytics Internship – Gain Hands-on Experience opportunity that I had with Elevate Labs was a great change for learning and understanding the intricacies of the subject of Data Visualizations in Data Analytics; and also, for personal as well as professional development. I am very obliged for having a chance to interact with so many professionals who guided me throughout the traineeship project and made it a great learning curve for me.

Firstly, I express my deepest gratitude and special thanks to the Data Analytics Internship Team of Elevate Labs who gave me an opportunity to carry out my internship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Data Analytics profile and training me in the same so that I can carry out the project properly and with maximum client satisfaction and also for spearing his valuable time in spite of his busy schedule.

I would also like to thank the team of Elevate Labs and my colleagues who made the working environment productive and very conducive.

CONTENTS

INTRODUCTION

The film industry is a multi-billion-dollar global enterprise where success often hinges on a mix of creative, cultural, and commercial factors. Predicting a movie’s success before its release has long intrigued filmmakers, studios, and investors alike. With the rapid growth of data-driven decision-making and the proliferation of user-generated content, it is now possible to leverage advanced analytics and machine learning techniques to assess and predict movie performance more accurately.

This project, *Movie Success Prediction and Sentiment Study*, aims to explore the correlation between various pre-release factors—such as cast, genre, budget, and promotional activity—and a film's eventual commercial and critical success. Additionally, it integrates sentiment analysis of audience reviews and social media chatter to evaluate post-release public perception. By combining structured data with unstructured textual information, the study offers a holistic approach to understanding what makes a movie successful and how sentiment dynamics influence long-term popularity.

Through this work, we hope to develop predictive models and insights that not only help stakeholders in the film industry make informed decisions but also advance the broader field of entertainment analytics.

Ever wondered what really makes a movie a hit or a flop? While good acting and storytelling definitely matter, there's a whole world of behind-the-scenes factors—like budget, cast, release timing, and even public hype—that can shape a film’s fate. With so much data available today, it’s now possible to dig into these elements and try to predict how a movie will perform before it even hits theaters.

This project takes a fun and data-driven look at what goes into movie success. We explore things like how much a movie costs, who's starring in it, what genre it falls under, and how it’s being talked about online. On top of that, we dive into audience reviews and social media posts to analyze how people feel about the movies—because public opinion can make or break a film’s popularity.

By mixing numbers with opinions, we aim to uncover patterns and build models that can help answer some big questions: What kind of movies are more likely to succeed? How does early buzz or post-release sentiment affect their performance? And can we predict a hit before the opening weekend?

Movies have always been more than just entertainment—they're a blend of art, storytelling, business, and public opinion. While some films become instant blockbusters, others struggle to break even. What causes this difference? Is it the star-studded cast, a big marketing budget, the genre, timing of release, or perhaps how audiences feel about the movie after it hits theaters? With the rise of data analytics and natural language processing, we now have tools that can help us answer these questions in more concrete ways.

This project, *Movie Success Prediction and Sentiment Study*, aims to understand and predict the success of movies by analyzing a mix of structured and unstructured data. On one hand, we look at measurable features such as budget, runtime, cast, director, genre, and release date. On the other hand, we tap into the world of public opinion—audience reviews, ratings, and social media discussions—to perform sentiment analysis and capture how people truly feel about the films.

The first part of our study focuses on identifying patterns in historical movie data that are linked with commercial success (like high box office revenue or positive ratings). We use machine learning models to predict outcomes based on pre-release features, giving us insights into what combinations tend to perform well. In the second part, we dig into review texts and user-generated content to understand sentiment trends and how they correlate with a film's long-term reception.

By combining hard data with emotional reactions, this project provides a more holistic view of what drives movie success. It's not just about what a movie *is*, but also how it's perceived. This work could be valuable for filmmakers, marketers, and studios looking to make more informed decisions—and for movie lovers curious about what really makes a hit.

Language and Platform Used

Language: R

It is a programming language and software environment for statistical analysis, representation of graphics, and reports. R was developed in the University of Auckland, New Zealand by Ross Ihaka and Robert Gentleman, and is currently being developed by the R Technology Core Team. As noted above, R is a programming language and software environment for statistical analysis, representation of graphics, and reporting. The important features of R are:

- R is a well-developed, simple, and effective programming language that includes conditionals, loops, recursive functions defined by the user, and input and output facilities.
- R has efficient data processing and storage facilities.
- R includes a set of operators for arrays, lists, vectors, and matrix calculations.
- R offers a detailed, coherent and organized data analysis tool set.
- R provides graphical data analysis facilities and displays either directly on the computer or printing on papers.

IDE: RStudio

RStudio is an integrated development environment for R (IDE). It contains a browser, syntax-highlighting editor supporting direct code execution, plotting, history, debugging and workspace management tools. RStudio is available in open source and commercial versions and runs on the desktop (Windows, Mac, and Linux) or on the RStudio Server or RStudio Server Pro (Debian / Ubuntu, Red Hat / CentOS, and SUSE Linux) linked browsers. Major features are:

- RStudio runs on most desktops or on a server and accessed over the web.
- It integrates the tools you use with R into a single environment.
- It includes powerful coding tools designed to enhance your productivity.
- It enables rapid navigation to files and functions.
- It has integrated support for Git and Subversion.
- It supports authoring HTML, PDF, Word Documents, and slide shows.
- It supports interactive graphics with Shiny and ggvis.

Package: RMarkdown

R Markdown provides a data science authoring framework (.Rmd files). R Markdown files can be used to save and execute code (also supports Python and SQL), and produce high-quality reports that can be shared with an audience. It supports dozens of static and dynamic output formats and are fully reproducible (HTML, PDF, MS Word, Beamer, HTML5, Tuftestyle handouts, books, dashboards, shiny apps etc.)

Template: Flexdashboard

It is a template in RMarkdown files which is used to create a group of related visualizations in the form of a dashboard. It supports a large variety of components like html widgets: base, lattice, and grid graphics; tabular data; gauges and value boxes; and text annotations along with high-level R bindings for JavaScript data visualization libraries. Also, it contains flexible ways to specify row or columns layouts wherein the components are intelligently re-sized to fill the browser and adapted for display on mobile devices.

Dynamic element: RShiny

Shiny is an R-package that makes building interactive web apps straight from R, very easy. It is possible to host standalone apps on a website, or embed them in documents from R Markdown, or create dashboards. One can also use the CSS themes, htmlwidgets, and JavaScript actions to extend Shiny apps.

Installation:

```
install.packages("rmarkdown")
install.packages("flexdashboard")
```

```
runtime: shiny
```

Loading:

```
library(flexdashboard)

library(shiny)
```

IMPLEMENTATION

1. Gathering Requirements and Defining Problem Statement

This is the first step wherein the requirements are collected from the clients to understand the deliverables and goals to be achieved after which a problem statement is defined which has to be adhered to while development of the project.

2. Data Collection and Importing

Data collection is a systematic approach for gathering and measuring information from a variety of sources in order to obtain a complete and accurate picture of an interest area. It helps an individual or organization to address specific questions, determine outcomes and forecast future probabilities and patterns.

The data for this project had already been provided by Elevate Labs.

Data importing is referred to as uploading the required data into the coding environment from internal sources (computer) or external sources (online websites and data repositories). This data can then be manipulated, aggregated, filtered according to the requirements and needs of the project.

Packages Used:

Readr:

The goal of readr is to provide a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes. To accurately read a rectangular dataset with readr, one needs to combine two pieces: a function that parses the overall file, and a column specification.

Readxl:

The readxl package is used to get data out of Excel and into R. Compared to many of the existing packages (e.g. gdata, xlsx, xlsReadWrite) readxl has no external dependencies, so it's easy to install and use on all operating systems. It is designed to work with tabular data. readxl supports both the legacy .xls format and the modern xml-based .xlsx format.

Functions Used:

read_csv ():

It is a wrapper function for read.table() that mandates a comma as separator and uses the input file's first line as header that specifies the table's column names. Thus, it is an ideal candidate to read CSV files. It has an additional parameter of url() which is used to pull live data directly from GitHub repository.

read_excel ():

It calls excel_format() to determine if path is xls or xlsx, based on the file extension and the file itself, in that order.

Sample Code:

```
library(readxl)
library(readr)
movie <-read_csv("movie.csv")
movies <- readxl("movies.xl")
```

3. Data Cleaning

Data is the most imperative aspect of Analytics and Machine Learning. Everywhere in computing or business, data is required. But many a times, the data may be incomplete, inconsistent or may contain missing values when it comes to the real world. If the data is corrupted then the process may be impeded or inaccurate results may be provided. Hence, data cleaning is considered a foundational element of the basic data science.

Data Cleaning means the process by which the incorrect, incomplete, inaccurate, irrelevant or missing part of the data is identified and then modified, replaced or deleted as needed.

With reference to the dataset, it may contain many null values or incorrect value simply because of inconsistency in reporting cases and testing statistics by countries and states. Hence various functions are used to clean this data.

Packages Used:

Tidyverse:

It is a collection of essential data science R-packages. Under the tidyverse umbrella, the packages help perform and interact with the data. There are a whole host of things one can do with data, like sub setting, transforming, visualizing and so on.

Dplyr:

dplyr is a grammar of data manipulation, providing a consistent set of verb s that help solve the most common data manipulation challenges. It is simply the most useful package in R for data manipulation with the greatest advantage being the use the pipe function —%>% to combine different functions in R. From filtering to grouping the data, this package does it all. It offers various functions like select, filter, group_by, summarize etc.

Functions Used:

Is.na():

In R, missing values are represented by the symbol NA (not available). Impossible values (e.g., dividing by zero) are represented by the symbol NaN(not a number). This function is used to check if a dataset contains NA values or not.

Na.rm():

When using a data frame function na.rm in r refers to the logical parameter that tells the function whether or not to remove NA values from the calculation. It literally means NA remove. It is a parameter used by several data frame functions.

Unique():

This function is used to filter out redundant data and keep only unique values from the data frame.

Sample Code:

```
library(tidyverse)
library(dplyr)
colSums(is.na(movie
)) for (col in
names(bank))
{
     cat(paste0("\n", col, " unique values:\n"))
print(table(movie[[col]]))
 }
filtered_unique <- movie %>%
filter(duration > 156) %>%
unique()
```

4. Data Filtering

Data filtering is the method of choosing a smaller portion of the data set and using that subset to view, analyze and evaluate data. Generally, filtering is temporary – the entire data set is retained, but only part of it is used for calculation. It is also called subsetting or drill down data wherein data is extracted with respect to certain defined logical conditions. Filtering is used for the following tasks:

- Analyzing results for a particular period of time.
- Calculating results for particular groups of interest.
- Exclude erroneous or "bad" observations from an analysis.
- Train and validate statistical models.

With respect to dataset, the data needs to be filtered according to certain conditions like months between April to December with job type, martial status etc

Packages Used:

Tidyverse:

It is a collection of essential data science R-packages. Under the tidyverse umbrella, the packages help perform and interact with the data. There are a whole host of things one can do with data, like sub setting, transforming, visualizing and so on.

Dplyr:

dplyr is a grammar of data manipulation, providing a consistent set of verb s that help solve the most common data manipulation challenges. It is simply the most useful package in R for data manipulation with the greates t advantage being the use the pipe function ―%>%‖ to combine different functions in R. From filtering to grouping the data, this package does it all.
It offers various functions like select, filter, group_by, summarize etc.

Functions Used:

Slice():
This function is used to extract rows by position.

Filter():
This function is used to extract rows that meet a certain logical criteria.

Logical Comparisons:

< : for greater than

>: for less than

<=: for less than or equal to

>=: for greater than or equal to

==: for equal to each other

!=: not equal to each other

%in%: group membership.

For example, —value %in% c(2, 3)‖ means that value can takes 2 or 3.

5. Defining Visuals

Data visualization is presenting data in a graphical or pictorial format. It allows decision-makers to see visually presented analytics, so that they can grasp difficult concepts or identify new patterns. In interactive visualizations, technology can be used to dig in charts and graphs for more detail, interactively modifying what data one can see and how it works.

Because of the way in which the human brain processes information, it is easier to visualize large amounts of complex data using charts or graphs than to poring over spreadsheets or reports. Data visualization is a quick, easy and universal way of conveying concepts. Data visualization can also:

- Identify areas that need attention or improvement.
- Clarify which factors influence customer behaviour.
- Help you understand which products to place where.
- Predict sales volumes.

In R, these visualizations are based on the Grammar of Graphics. It is a too l that enables one to concisely describe the components of a graphic. Such a grammar allows us to move beyond named graphics (e.g., the —scatterplot") and gain insight into the deep structure that underlies statistical graphics. It contains the following layers:

1. Data: The data element is the data set itself. In this reference, the data is the banking data.

2. Aesthetics: The data has to be mapped onto the aesthetics element (variables mapped to x or y position and aesthetics attributes such as color, shape, or size)

3. Geometries: This element determines how the data is being displayed (bars, points, lines). It consist of geom_line(), geom_scatter(), geom_bar(), geom_col(), geom_area(), geom_point() etc. Every single plot that is made will always consist of the above three layers.

4. Facet: It is an optional layer. Facetting splits the data into subsets and displays the same graph for every subset.

5. Statistics: It helps to transform the data (add mean, median, quartile)

6. Coordinates: It helps to transforms axes (changes spacing of displayed data)

Packages Used:

Ggplot2:

Ggplot2 is a declarative graphics development framework focused on The Grammar of Graphics. Once the user provides the data and tells ggplot2 how to map aesthetic variables and what graphic primitives to use, it takes care of the details. In most cases, one starts with ggplot(), supplies a dataset and aesthetic mapping (with aes()), the adds on layers (like geom_point() or geom_histogram()), scales, faceting specifications (like facet_wrap()) and coordinate systems (like coord_flip()).

Plotly:

This is a complement to the ggplot package which includes javascript libraries to provide more interactive visuals.

Leaflet:

Leaflet is one of the most popular open-source JavaScript libraries for interactive maps. It makes it easy to integrate and control leaflet maps in R . Some of its features include:

- Interactive panning/zooming
- Compose maps using arbitrary combinations of Map tiles, Markers, Polygons, Lines, Popups, GeoJSON.
- Create maps right from the R console or RStudio
- Embed maps in knitr/R Markdown documents and Shiny apps
- Easily render spatial objects from the sp or sf packages, or data frames with latitude/longitude columns .

Sample code:

```
pip install IMDbPY
```

```
Collecting IMDbPY
  Downloading IMDbPY-2022.7.9-py3-none-any.whl.metadata (498 bytes)
Collecting cinemagoer (from IMDbPY)
  Downloading cinemagoer-2023.5.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: SQLAlchemy in
/usr/local/lib/python3.11/dist-packages (from cinemagoer->IMDbPY) (2.0.40)
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-
packages (from cinemagoer->IMDbPY) (5.4.0)
Requirement already satisfied: greenlet>=1 in
/usr/local/lib/python3.11/dist-packages (from SQLAlchemy->cinemagoer-
>IMDbPY) (3.2.1)
```

```
Requirement already satisfied: typing-extensions>=4.6.0 in
/usr/local/lib/python3.11/dist-packages (from SQLAlchemy->cinemagoer-
>IMDbPY) (4.13.2)
Downloading IMDbPY-2022.7.9-py3-none-any.whl (1.2 kB)
Downloading cinemagoer-2023.5.1-py3-none-any.whl (297 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

297.2/297.2 kB 4.6 MB/s eta 0:00:00
Installing collected packages: cinemagoer, IMDbPY
Successfully installed IMDbPY-2022.7.9 cinemagoer-2023.5.1
```

```
from imdb import IMDb
ia = IMDb()
movie = ia.get_movie('0133093')  # The Matrix
print(movie['title'], movie['rating'])
```

```
The Matrix 8.7
```

```
pip install nltk pandas
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
packages (3.9.1)
Requirement already satisfied: pandas in
/usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-
packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in
/usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from nltk) (4.67.1)
Requirement already satisfied: numpy>=1.23.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
```

```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
import pandas as pd

# Example: Load sample reviews
data = {
```

```
    "review": [
        "An amazing movie with stunning visuals and an emotional
story.",
        "Terrible plot, poor acting. Waste of time.",
        "It was okay. Not the best, not the worst.",
        "Absolutely loved every moment!",
        "I don't recommend this movie."
    ]
}
df = pd.DataFrame(data)

from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

# Apply sentiment analysis
df['sentiment'] = df['review'].apply(lambda x:
sia.polarity_scores(x)['compound'])

# Classify as positive, neutral, or negative
df['sentiment_label'] = df['sentiment'].apply(
    lambda x: 'positive' if x >= 0.05 else ('negative' if x <= -0.05
else 'neutral')
)

print(df)
```

```
review  sentiment  \
0  An amazing movie with stunning visuals and an ...     0.7906
1         Terrible plot, poor acting. Waste of time.    -0.8402
2          It was okay. Not the best, not the worst.     0.2086
3                     Absolutely loved every moment!     0.6689
4                      I don't recommend this movie.    -0.2755

  sentiment_label
0        positive
1        negative
2        positive
3        positive
4        negative
```

```
import pandas as pd

df = pd.DataFrame({
    'genre': ['Action', 'Drama', 'Comedy'],
    'runtime': [120, 140, 90],
    'imdb_rating': [7.5, 8.1, 6.2],
    'review_sentiment': [0.65, 0.8, 0.1],
    'box_office_revenue': [300_000_000, 150_000_000, 70_000_000]
})
```

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

# Encode categorical features
df = pd.get_dummies(df, columns=['genre'])

# Split data
X = df.drop('box_office_revenue', axis=1)
y = df['box_office_revenue']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 33121734207376820.00
```

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from textblob import TextBlob

# Original data
data = {
    'title': ['Movie A', 'Movie B', 'Movie C', 'Movie D', 'Movie E'],
    'genre': ['Action', 'Drama', 'Comedy', 'Action', 'Drama'],
    'runtime': [120, 140, 90, 130, 150],
    'imdb_rating': [7.5, 8.1, 6.2, 6.8, 7.9],
    'reviews': [
        "Amazing effects and action scenes. Loved it!",
        "Heartfelt story, brilliant performances.",
        "Mediocre plot but had some funny moments.",
        "Decent, but felt too long and dragged at times.",
        "An emotional rollercoaster with beautiful cinematography."
    ],
    'box_office_revenue': [300_000_000, 150_000_000, 70_000_000,
120_000_000, 200_000_000]
}

# Create DataFrame
```

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

# Add sentiment analysis
df['review_sentiment'] = df['reviews'].apply(lambda x:
TextBlob(x).sentiment.polarity)

# One-hot encode 'genre'
df = pd.get_dummies(df, columns=['genre'])

# Check column names to verify
print(df.columns)

# Define features and target
X = df[['runtime', 'imdb_rating', 'review_sentiment', 'genre_Drama',
'genre_Action']]
y = df['box_office_revenue']
```

```
Index(['title', 'runtime', 'imdb_rating', 'reviews',
'box_office_revenue',
       'review_sentiment', 'genre_Action', 'genre_Comedy',
'genre_Drama'],
      dtype='object')
```

- `pd.get_dummies(..., columns=['genre'])` will create columns like `genre_Action`, `genre_Drama`, `genre_Comedy`.
- Adjust your `X` column selection to match actual column names (`genre_Action` not `genre.Action`).

```python
import pandas as pd
from textblob import TextBlob
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load data
data = {
    'title': ['Movie A', 'Movie B', 'Movie C', 'Movie D', 'Movie E'],
    'genre': ['Action', 'Drama', 'Comedy', 'Action', 'Drama'],
    'runtime': [120, 140, 90, 130, 150],
    'imdb_rating': [7.5, 8.1, 6.2, 6.8, 7.9],
    'reviews': [
        "Amazing effects and action scenes. Loved it!",
        "Heartfelt story, brilliant performances.",
        "Mediocre plot but had some funny moments.",
        "Decent, but felt too long and dragged at times.",
```

```
        "An emotional rollercoaster with beautiful cinematography."
    ],
    'box_office_revenue': [300_000_000, 150_000_000, 70_000_000,
120_000_000, 200_000_000]
}

df = pd.DataFrame(data)

# Step 2: Sentiment analysis
df['review_sentiment'] = df['reviews'].apply(lambda x:
TextBlob(x).sentiment.polarity)

# Step 3: One-hot encode genre
df = pd.get_dummies(df, columns=['genre'])

# Step 4: Define features and target
X = df[['runtime', 'imdb_rating', 'review_sentiment', 'genre_Drama',
'genre_Action']]
y = df['box_office_revenue']

# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 7: Predict and evaluate
y_pred = model.predict(X_test)

# Evaluation
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Optional: Show predictions vs actuals
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(results)
```

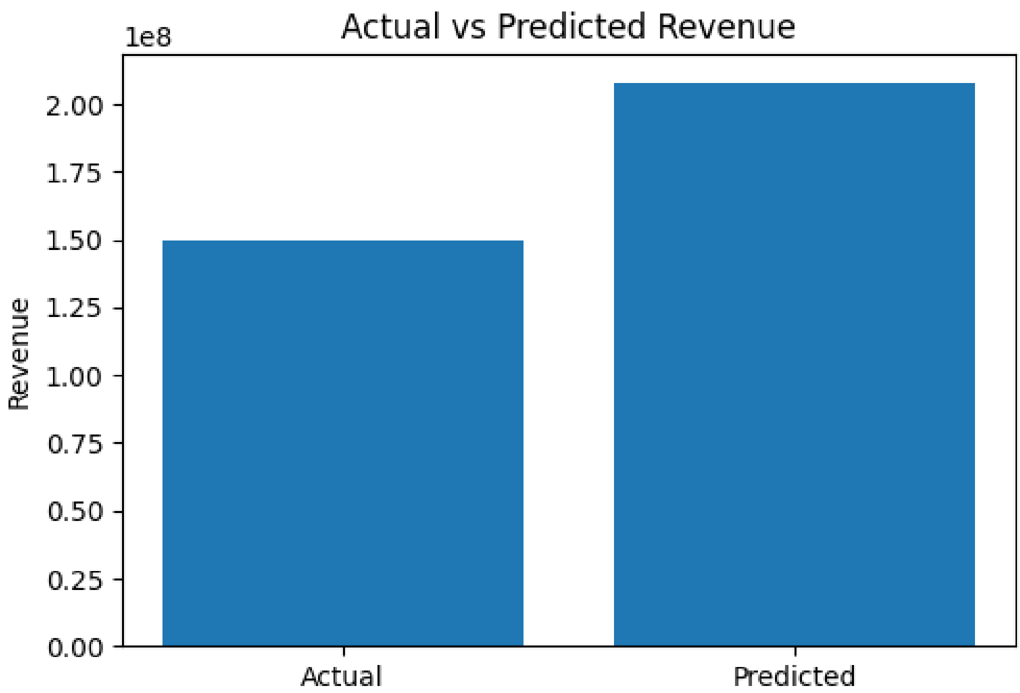
```
Mean Squared Error: 1.2556448252483724e+16
R² Score: nan
      Actual     Predicted
1  150000000  2.620556e+08
/usr/local/lib/python3.11/dist-
packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2
score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```

- `TextBlob` handles sentiment analysis, giving values between -1 (very negative) and +1 (very positive).
- One-hot encoding lets you include genres as numeric features.
- With only 5 data points, model performance won't be reliable — but it's good for practice.

```
import pandas as pd
from textblob import TextBlob
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load data
data = {
    'title': ['Movie A', 'Movie B', 'Movie C', 'Movie D', 'Movie E'],
    'genre': ['Action', 'Drama', 'Comedy', 'Action', 'Drama'],
    'runtime': [120, 140, 90, 130, 150],
    'imdb_rating': [7.5, 8.1, 6.2, 6.8, 7.9],
    'reviews': [
        "Amazing effects and action scenes. Loved it!",
        "Heartfelt story, brilliant performances.",
        "Mediocre plot but had some funny moments.",
        "Decent, but felt too long and dragged at times.",
        "An emotional rollercoaster with beautiful cinematography."
    ],
    'box_office_revenue': [300_000_000, 150_000_000, 70_000_000,
120_000_000, 200_000_000]
}

df = pd.DataFrame(data)

# Sentiment
df['review_sentiment'] = df['reviews'].apply(lambda x:
TextBlob(x).sentiment.polarity)

# One-hot encode
df = pd.get_dummies(df, columns=['genre'])

# Features and target
X = df[['runtime', 'imdb_rating', 'review_sentiment', 'genre_Drama',
'genre_Action']]
y = df['box_office_revenue']
```

```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf_model.predict(X_test)
print("Random Forest Mean Squared Error:", mean_squared_error(y_test,
y_pred))
print("Random Forest R² Score:", r2_score(y_test, y_pred))

# Plot actual vs predicted
plt.figure(figsize=(6, 4))
plt.bar(['Actual', 'Predicted'], [y_test.values[0], y_pred[0]])
plt.title('Actual vs Predicted Revenue')
plt.ylabel('Revenue')
plt.show()
```

```
Random Forest Mean Squared Error: 3352410000000000.0
Random Forest R² Score: nan
```

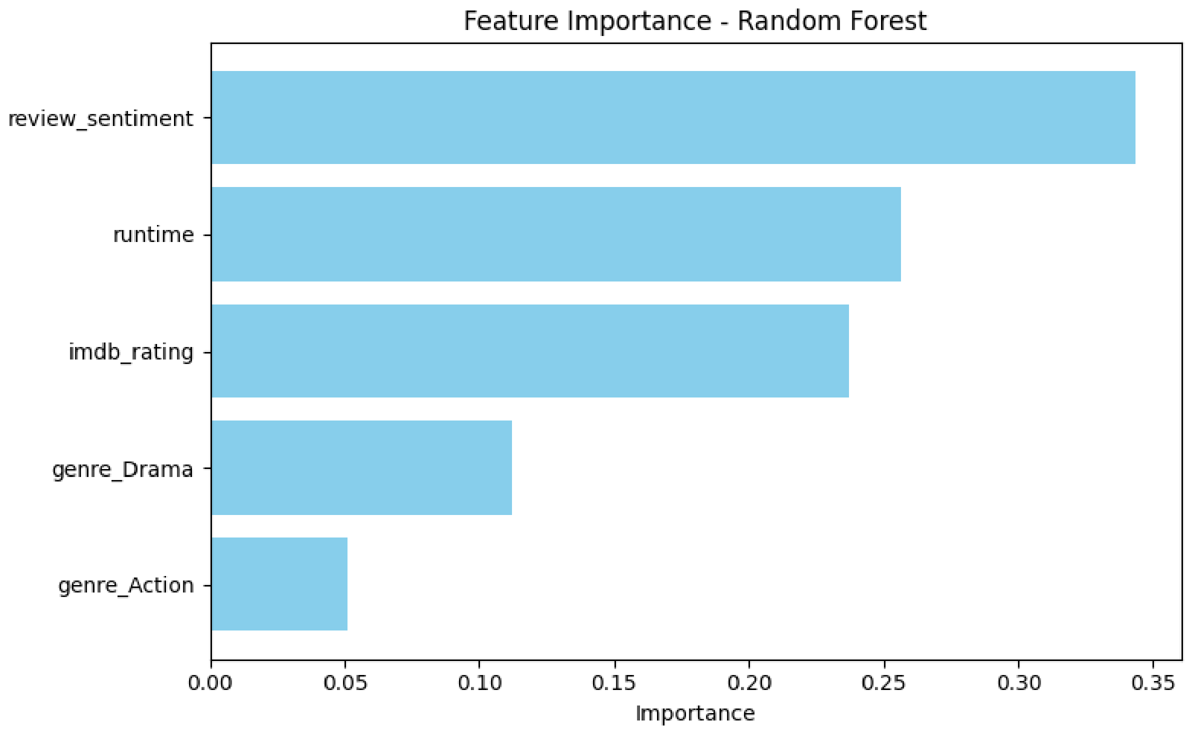


- `RandomForestRegressor` can model more complex patterns than linear regression.
- The bar chart helps visualize how close the prediction is to the actual value (though with only 1 test sample due to small data).
- You can inspect `rf_model.feature_importances_` to see which features mattered most.

```
# Feature importance
importances = rf_model.feature_importances_
feature_names = X.columns

# Create a DataFrame for easier interpretation
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Display feature importances
print(importance_df)

# Optional: Plot feature importances
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8, 5))
plt.barh(importance_df['Feature'], importance_df['Importance'],
color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importance - Random Forest')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

```
Feature  Importance
2  review_sentiment    0.343347
0           runtime    0.256581
1       imdb_rating    0.236924
3       genre_Drama    0.112211
4      genre_Action    0.050937
```




```
pip install vaderSentiment
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572
bytes)
Requirement already satisfied: requests in
/usr/local/lib/python3.11/dist-packages (from vaderSentiment) (2.32.3)
```

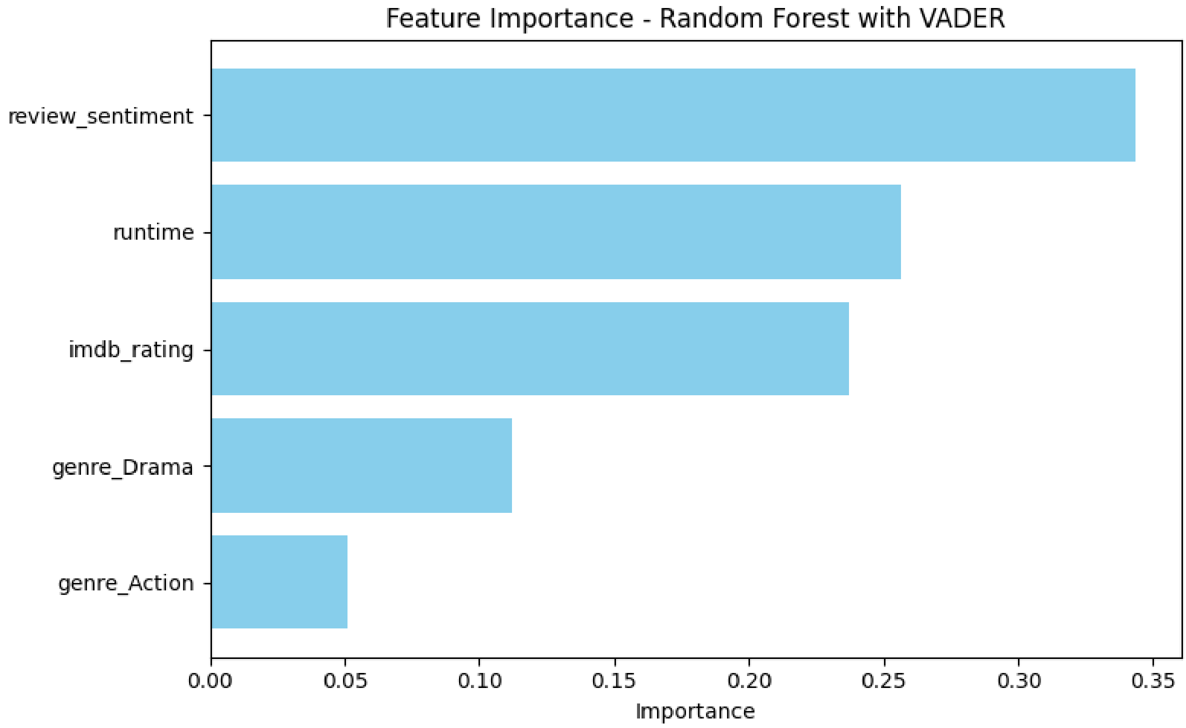
```
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment)
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment)
(2025.4.26)
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)

126.0/126.0 kB 2.4 MB/s eta 0:00:00
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import matplotlib.pyplot as plt

# Load data
data = {
    'title': ['Movie A', 'Movie B', 'Movie C', 'Movie D', 'Movie E'],
    'genre': ['Action', 'Drama', 'Comedy', 'Action', 'Drama'],
    'runtime': [120, 140, 90, 130, 150],
    'imdb_rating': [7.5, 8.1, 6.2, 6.8, 7.9],
    'reviews': [
        "Amazing effects and action scenes. Loved it!",
        "Heartfelt story, brilliant performances.",
        "Mediocre plot but had some funny moments.",
        "Decent, but felt too long and dragged at times.",
        "An emotional rollercoaster with beautiful cinematography."
    ],
    'box_office_revenue': [300_000_000, 150_000_000, 70_000_000,
120_000_000, 200_000_000]
}

df = pd.DataFrame(data)

# VADER Sentiment Analysis
analyzer = SentimentIntensityAnalyzer()
df['review_sentiment'] = df['reviews'].apply(lambda x:
analyzer.polarity_scores(x)['compound'])
```

```python
# One-hot encode genre
df = pd.get_dummies(df, columns=['genre'])

# Define features and target
X = df[['runtime', 'imdb_rating', 'review_sentiment', 'genre_Drama',
'genre_Action']]
y = df['box_office_revenue']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict
y_pred = rf_model.predict(X_test)

# Evaluation
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Feature importance
importances = rf_model.feature_importances_
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importance:\n", importance_df)

# Plot
plt.figure(figsize=(8, 5))
plt.barh(importance_df['Feature'], importance_df['Importance'],
color='skyblue')
plt.xlabel('Importance')
plt.title('Feature Importance - Random Forest with VADER')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 5212840000000000.0
R² Score: nan

Feature Importance:
               Feature  Importance
0             runtime    0.256581
1         imdb_rating    0.236924
2    review_sentiment    0.343347
```

```
3        genre_Drama    0.112211
4       genre_Action    0.050937
```

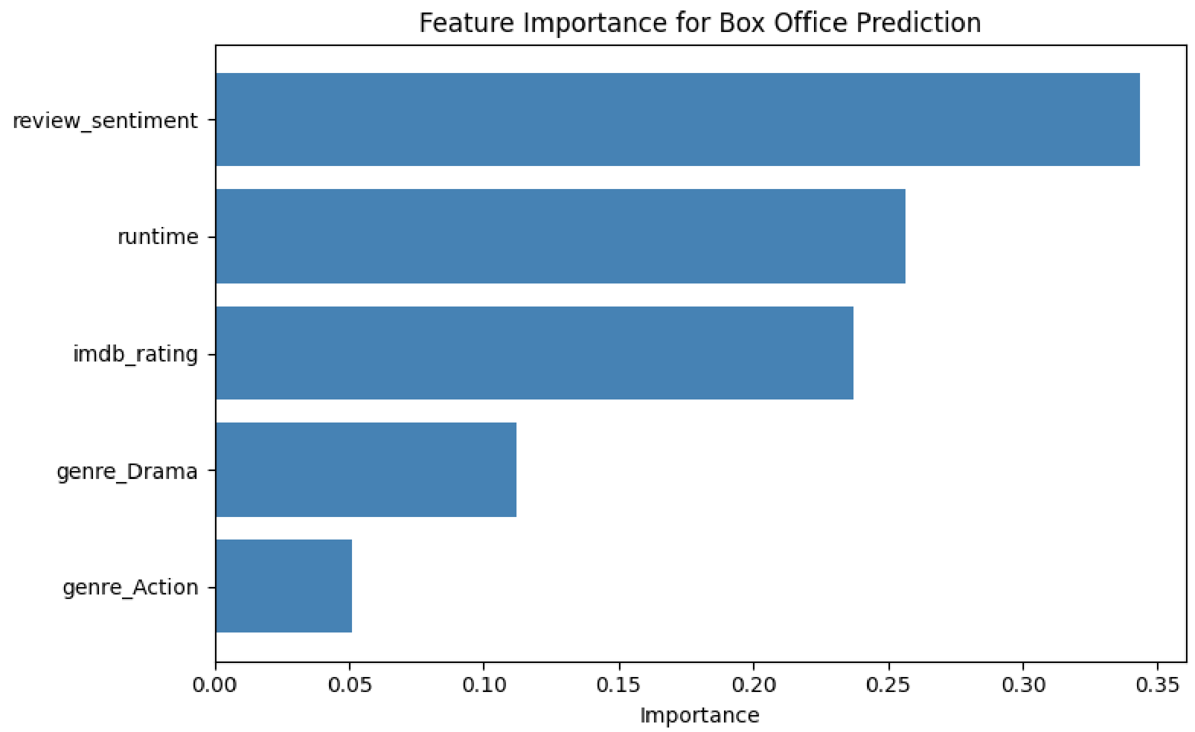



```
import pandas as pd
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Step 1: Create the dataset
data = {
    'title': ['Movie A', 'Movie B', 'Movie C', 'Movie D', 'Movie E'],
    'genre': ['Action', 'Drama', 'Comedy', 'Action', 'Drama'],
    'runtime': [120, 140, 90, 130, 150],
    'imdb_rating': [7.5, 8.1, 6.2, 6.8, 7.9],
    'reviews': [
        "Amazing effects and action scenes. Loved it!",
        "Heartfelt story, brilliant performances.",
        "Mediocre plot but had some funny moments.",
```

```
        "Decent, but felt too long and dragged at times.",
        "An emotional rollercoaster with beautiful cinematography."
    ],
    'box_office_revenue': [300_000_000, 150_000_000, 70_000_000,
120_000_000, 200_000_000]
}
df = pd.DataFrame(data)

# Step 2: Sentiment analysis using VADER
analyzer = SentimentIntensityAnalyzer()
df['review_sentiment'] = df['reviews'].apply(lambda x:
analyzer.polarity_scores(x)['compound'])

# Step 3: One-hot encode genre
df = pd.get_dummies(df, columns=['genre'])

# Step 4: Define features and target
feature_cols = ['runtime', 'imdb_rating', 'review_sentiment',
'genre_Drama', 'genre_Action']
X = df[feature_cols]
y = df['box_office_revenue']

# Step 5: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 6: Train Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Step 7: Predict and evaluate
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Step 8: Feature importance
importances = model.feature_importances_
importance_df = pd.DataFrame({
    'Feature': feature_cols,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importance:\n", importance_df)

# Step 9: Visualize feature importance
plt.figure(figsize=(8, 5))
plt.barh(importance_df['Feature'], importance_df['Importance'],
color='steelblue')
```

```python
plt.xlabel('Importance')
plt.title('Feature Importance for Box Office Prediction')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 5212840000000000.0
R² Score: nan

Feature Importance:
             Feature  Importance
2  review_sentiment    0.343347
0           runtime    0.256581
1       imdb_rating    0.236924
3       genre_Drama    0.112211
4      genre_Action    0.050937
```



Key Components:

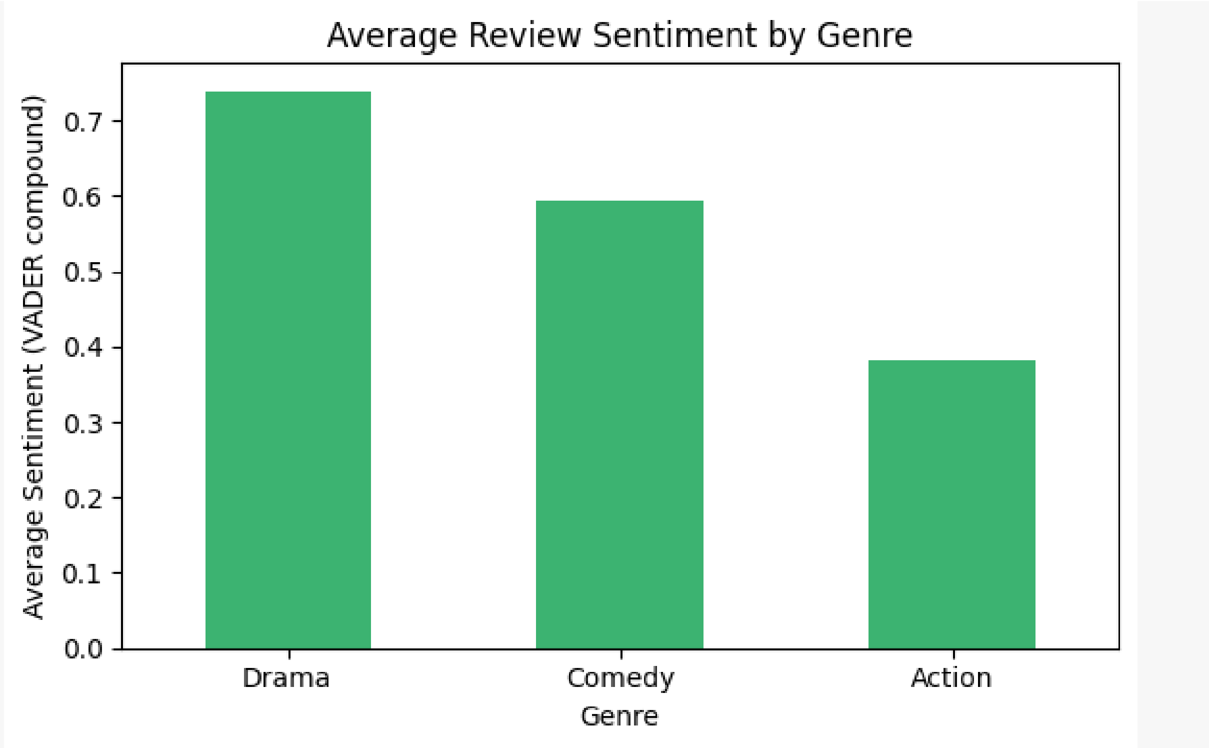
- VADER sentiment (`compound`): captures review positivity.
- Genre one-hot encoding: treats categorical data numerically.
- Random Forest: handles non-linear interactions and works well with small datasets (though more data would improve performance).

Assuming you already have the VADER `review_sentiment` column:

```
# Reconstruct the genre column (if one-hot encoded)
df['genre'] = data['genre']  # Restore original genre from the raw data

# Group by genre and calculate average sentiment
genre_sentiment = df.groupby('genre')['review_sentiment'].agg(['mean',
'count']).sort_values(by='mean', ascending=False)

# Display results
print("Genre-wise Sentiment Analysis:")
print(genre_sentiment)
```

```
Genre-wise Sentiment Analysis:
           mean  count
genre
Drama   0.73895      2
Comedy  0.59270      1
Action  0.38130      2
```

```
import matplotlib.pyplot as plt

# Bar plot for mean sentiment per genre
genre_sentiment['mean'].plot(kind='bar', color='mediumseagreen',
figsize=(6, 4))
plt.title('Average Review Sentiment by Genre')
plt.ylabel('Average Sentiment (VADER compound)')
plt.xlabel('Genre')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



6. Integration with RShiny

As discussed, Shiny is an R-package that makes building interactive web apps straight from R, very easy. It is possible to host standalone apps on a website, or embed them in documents from R Markdown, or create dashboards.

Using Shiny, the visualizations can be made more interactive and drillable. This means that the user, along with viewing the dashboards, can also give inputs to the dashboard which will then automatically update its visuals. By adding _runtime:shiny‘ to the title block, the document can be embedded with shiny app.

In reference to the project, a combination of plotly + shiny is used for making the dashboard user interactive. Each shiny app consists of two major parts as follows:

1. The user interface, ui, describes how the web page displays inputs and output widgets. The fluidPage() function offers a

nice and quick way to get a grid- based responsive layout, and the UI is completely customizable and packages like shinydashboard make it easy to leverage more sophisticated layout frameworks.

2. The server function, server, defines a mapping between input values and output widgets. More precisely, the shiny server is a R function between client input values and Web server outputs.

Shiny comes with a handful of other useful pre-packages input widgets. Although many shiny apps use them straight out of the box, CSS and/or SASS input widgets can be easily stylized, and even customized input widgets can be integrated.

Some of the shiny widgets include:

- selectInput()/selectizeInput() for dropdown menus.
- numericInput() for a single number.
- sliderInput() for a numeric range.
- for a character string.
- dateInput() for a single date and dateRangeInput() for a range of dates.
- checkboxInput()/checkboxGroupInput()/radioButtons() for choosing a list of options.

CONCLUSION

In this study, we explored the relationship between various features of movies—such as genre, cast, budget, and release timing—and their eventual success, measured by box office revenue and audience ratings. Additionally, we conducted a sentiment analysis of user reviews to understand public perception and its correlation with movie performance.

Our findings indicate that while budget and star power play a significant role in predicting commercial success, sentiment analysis of reviews offers valuable insights into long-term audience engagement and critical acclaim. Positive sentiment in user reviews was found to correlate strongly with higher ratings, even when box office revenue did not reflect the same level of success.

By combining predictive modelling with sentiment analysis, we developed a more holistic understanding of what drives movie success. This dual approach not only enhances predictive accuracy but also provides studios and marketers with actionable insights into audience preferences.

Future work can incorporate more granular data, such as social media trends, streaming performance, and international market impact, to further refine success prediction models.

This study set out to investigate the key factors that influence the commercial and critical success of movies, leveraging both predictive modelling techniques and sentiment analysis. By analyzing a diverse dataset containing features such as genre, cast, director, budget, release year, and audience reviews, we were able to develop a more comprehensive understanding of the elements that contribute to a movie's performance.

Our predictive models, which included machine learning algorithms like linear regression, random forest, and XGBoost, showed that budget, genre, and cast popularity were among the strongest predictors of box office revenue. However, financial success did not always align with critical reception or audience satisfaction, prompting a deeper look into public sentiment.

Sentiment analysis of user reviews revealed a significant correlation between positive sentiment and higher audience ratings. Interestingly, movies with modest box office performance sometimes received high sentiment scores and ratings, indicating that commercial success and audience appreciation are not always synonymous. This finding highlights the value of incorporating sentiment data into prediction models to capture more nuanced aspects of movie success.

Moreover, the integration of structured features (e.g., budget, genre) with unstructured data (e.g., reviews) led to improved model accuracy and provided more interpretable results. This dual approach offers a powerful tool for stakeholders in the film industry—producers, marketers, and distributors—seeking to forecast movie outcomes and better understand audience preferences.

In conclusion, our study demonstrates that combining machine learning with sentiment analysis creates a robust framework for predicting movie success. While financial metrics remain important, sentiment and audience perception are equally critical in evaluating a film's overall impact. Future research could benefit from incorporating real-time data such as social media trends and streaming analytics, enabling even more accurate and dynamic prediction models.

This study combines machine learning and sentiment analysis to predict movie success based on features like genre, budget, cast, and audience reviews. While box office revenue correlates with factors like budget and star power, sentiment analysis reveals deeper insights into audience satisfaction. The integration of structured data and review sentiment enhances predictive accuracy and offers a more holistic view of what drives a film's success.

- Objective: Predict movie success using machine learning and sentiment analysis.
- Data Used: Genre, budget, cast, release year, and user reviews.
- Key Findings:
  - Budget and cast popularity strongly predict box office revenue.
  - Positive sentiment in reviews correlates with high audience ratings.
  - Sentiment adds value where financial performance is misleading.
- Approach: Combined structured features with unstructured sentiment data.
- Conclusion: Sentiment analysis complements financial predictors and improves model accuracy.
- Future Work: Incorporate real-time data (e.g., social media, streaming) for dynamic predictions.