Yunnan University Software College Big Data Analytics & Application Report

School of Software, Yunnan University

Personal Grades

(Students fill in (except the last column))

Serial number	Student number	Name	Major	Score
1	20233120001	MIA MD RAKIB	AI	

Semester: Fall 2025

Course Name: Big Data Analytics & Application

Teacher: Tang li

Project Name: Analyze the movie dataset using PySpark

Team Leader: MIA MD RAKIB

Report completion Date: 2025-12-30

Big Data Analytics & Application (Elementary) Autumn 2025
Final Course Report Requirements

A. Introduction/ Situation/ Background Information

This project evaluates the students' understanding of Spark, MLlib and machine learning algorithm.

B. Course Learning Outcomes (CLO) covered

At the end of this assessment, students are able to:

CLO 1	Describe the concept and understand broad knowledge of Big Data.
CLO 2	Demonstrate appropriate knowledge of Hadoop, Spark, Storm and Map Reduce framework and apply them to build a VM-based environment.
CLO 3	Integrate knowledge and understanding of the basic principles, techniques and methodologies of organizing and searching Big Data, and apply them to create value with business insight.
CLO 4	Identify the awareness of the wide applicability of Big Data for real-world practical purposes.
CLO 5	Demonstrate the need to continually follow Big Data development trends.

C. University Policy on Academic Misconduct

1. Academic misconduct is a serious offense in Xiamen University Malaysia. It can be defined as any of the following:
    - i. **Plagiarism** is submitting or presenting someone else's work, words, ideas, data or information as your own intentionally or unintentionally. This includes incorporating published and unpublished material, whether in manuscript, printed or electronic form into your work without acknowledging the source (the person and the work).
    - ii. **Collusion** is two or more people collaborating on a piece of work (in part or whole) which is intended to be wholly individual and passed it off as own individual work.
    - iii. **Cheating** is an act of dishonesty or fraud in order to gain an unfair advantage in an assessment. This includes using or attempting to use, or assisting another

to use materials that are prohibited or inappropriate, commissioning work from a third party, falsifying data, or breaching any examination rules.

2. All the assessment submitted must be the outcome of the student. Any form of academic misconduct is a serious offense which will be penalised by being given a zero mark for the entire assessment in question or part of the assessment in question. If there is more than one guilty party as in the case of collusion, both you and your collusion partner(s) will be subjected to the same penalty.

D. Instruction to Students

This is an **individual** project. Each student should submit:

1. A zip file containing:
    a. A Jupyter Notebook file, named as "Project_YourStudentID.ipynb". All the codes, analysis, figures should be included in a single file with nice and neat format. Use Markdown cell with different headers for different sections.
    b. The Python source code of all your implementation. Just export all of your codes from Jupyter Notebook in a .py file, named as "Project_YourStudentID.py". This is for plagiarism detection.
    c. PowerPoint slides.
2. The course report must be printed (or written) on A4 paper in the following order and bound into a booklet:

    a) A standardized cover page;
    b) Requirements for the final course report;
    c) Project members and their assigned tasks;
    d) Group performance evaluation form for the final course report;
    e) Individual performance evaluation form for the final course report (one per person);
    f) Title, author(s), abstract, and keywords;
    g) Table of contents/outline;
    h) Main body of the final report, which may be divided into sections and should include the following content:

        (1) Project Background

(2) Algorithm Principle
(3) Model Design
(4) Experimental Results and Analysis、
(5) Conclusion、
(6) References.

The deadline of Project is **18:00, 3th Jan 2026**. Overdue penalty will be given to the project that is submitted after the deadline.

* Your codes will be sent to a **Plagiarism** detection system for duplication checking. Please write your codes independently. (Modify your code if you copy some fragment from the Internet because your classmates may copy the same fragment.)

E. Evaluation Breakdown

No.	**Component Title**	**Percentage (%)**
1	Project presentation	40
2	Project report	60
	TOTAL	**100**

F. Task(s)

You are required to analyze the Movie Dataset using PySpark. The dataset can be downloaded on Moodle. These files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

This dataset consists of the following files:
movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.
keywords.csv: Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.

credits.csv: Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

links.csv: The file that contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset. You may obtain more information by using a crawler program with the TMDB and IMDB IDs:

- imdbId is an identifier for movies used by http://www.imdb.com. E.g., the movie Toy Story has the link http://www.imdb.com/title/tt0114709/.
- tmdbId is an identifier for movies used by https://www.themoviedb.org. E.g., the movie Toy Story has the link https://www.themoviedb.org/movie/862.

ratings.csv: This file contains 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.

Questions and tasks:

1. Build regression models to predict movie revenue and vote averages based on a certain metric. (40 marks)
2. Analyze that what movies tend to get higher vote averages on TMDB. Try to use more figures with data visualization methods to illustrate your analysis. (20 marks)
3. Use collaborative filtering to build a movie recommendation system with two functions:
    a. Suggest top N movies similar to a given movie title (20 marks).
    b. Predict user rating for the movies they have not rated for. (20 marks).

Show all steps including data preprocessing, modeling, testing, evaluations with concise explanation in Markdown cell. You may also try different models and compare them in different ways with discussion. If your personal computer is not powerful enough to handle this project, you may try to use some public computation resources like Google Colab.

Table of Contents

Abstract

This study presents a comprehensive, large-scale data analysis and predictive modeling workflow applied to a MovieLens-derived dataset. Leveraging the distributed computing capabilities of Apache Spark, implemented via PySpark, the research addresses two critical prediction tasks: forecasting movie revenue and predicting average user ratings. The methodology encompasses rigorous data ingestion, cleaning, and feature engineering, followed by the implementation and comparative evaluation of three distinct machine learning models: Linear Regression, Random Forest Regressor, and Gradient Boosting Regressor. The results indicate that while financial metrics like revenue are highly predictable from pre-release features (with the Gradient Boosting model achieving an R^2 of 0.7377), the prediction of subjective user ratings remains a significant challenge (best R^2 of 0.4418). This work not only validates the efficacy of ensemble methods for non-linear prediction tasks in the film domain but also provides a detailed, scalable blueprint for big data analysis in recommender systems research, building upon the foundational context established by the creators of the MovieLens platform.

Keywords

PySpark, MovieLens, Big Data Analytics, Predictive Modeling, Regression Analysis, Gradient Boosting, Random Forest, Revenue Prediction, Vote Average Prediction, Collaborative Filtering, Recommender Systems, Machine Learning, Distributed Computing, Data Visualization, Feature Engineering

1. Introduction

The digital age has ushered in an era of information overload, particularly within the entertainment industry. Consumers are faced with an ever-expanding catalog of movies, music, and media, making the task of selection increasingly complex. Recommender Systems (RS) have emerged as an indispensable technology to mitigate this complexity, acting as intelligent filters that guide users toward items of interest [3]. The MovieLens dataset, curated by the GroupLens research group at the University of Minnesota, stands as the most influential and widely adopted benchmark for collaborative filtering and RS research globally [2].

This research is motivated by the dual challenge of understanding and predicting movie success. Success in the film industry is multifaceted, encompassing both **financial performance** (revenue) and **critical reception/user satisfaction** (average rating). Predicting these outcomes is of immense value to producers, distributors, and streaming platforms for strategic planning, resource allocation, and content acquisition.

The primary objectives of this study are:

1 **Scalable Data Processing:** To establish a robust and efficient data pipeline using PySpark for the ingestion, cleaning, and feature engineering of a large, multi-source movie dataset.

2 **Exploratory Data Analysis (EDA):** To conduct a detailed EDA to uncover the statistical distributions, correlations, and underlying patterns between key movie features, such as budget, popularity, and user engagement metrics.

3 **Predictive Modeling:** To develop and comparatively evaluate machine learning models for two distinct regression tasks:

- **Revenue Prediction:** Forecasting a movie's box office revenue based on its intrinsic and pre-release features.
- **Vote Average Prediction:** Forecasting the average user rating, a proxy for critical success and user satisfaction.

The subsequent sections of this report will detail the theoretical context of the MovieLens dataset, the technical methodology employed, the quantitative results of the predictive models, and a discussion of the findings in the context of existing literature.

2. Related Work

The current study is situated at the intersection of three major academic domains: Recommender Systems and Collaborative Filtering, Movie Success Prediction, and Large-Scale Data Processing.

2.1. Recommender Systems and the MovieLens Legacy

The field of Recommender Systems is fundamentally concerned with predicting a user's preference for an item [4]. **Collaborative Filtering (CF)**, the dominant paradigm in this field, operates on the principle that users who agreed in the past will agree again in the future [5]. The MovieLens system, from its inception in 1997, has been a living laboratory for CF research [2].

The papers provided by the user offer crucial historical and contextual grounding:

> ***"MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System" (Miller et al., 2003)*** *[1] documents an early effort to extend the MovieLens service to mobile devices (PDAs) using the AvantGo service. This work highlights the practical challenges of deploying RS in resource-constrained, occasionally connected environments, emphasizing the trade-off between data synchronization depth and user experience. It underscores the importance of interface design and the need for offline functionality, which are key considerations for any real-world RS deployment.*

> ***"The MovieLens Datasets: History and Context" (Harper & Konstan, 2015)*** *[2] provides an essential retrospective on the evolution of the MovieLens platform and its resulting datasets. The authors detail how changes in the user interface, rating scales, and data collection methods over nearly two decades have subtly influenced the data's characteristics. This context is vital for any researcher using the dataset, as it cautions against treating the data as a static, monolithic entity. The paper implicitly encourages researchers to move beyond simple CF to explore the rich metadata (like budget, revenue, and genres) that accompany the ratings, which is precisely the focus of the current predictive modeling effort.*

The evaluation of CF systems is a mature area, with seminal work by Herlocker et al. [3] establishing the distinction between **predictive accuracy** (e.g., RMSE, MAE) and **decision support accuracy** (e.g., precision, recall). While the current study focuses on predictive accuracy metrics (RMSE, R^2, MAE) for the regression tasks, the underlying goal is to inform decision support—specifically, the decision of which movies are likely to be financially or critically successful.

2.2. Predictive Modeling of Movie Success

Predicting movie success, whether measured by box office revenue or critical acclaim, has been a long-standing challenge in data science. Early models often relied on simple linear regression, but modern approaches utilize more sophisticated machine learning techniques to capture the complex, non-linear interactions between features [8].

Research consistently identifies several key feature categories as predictive of success:

- **Financial Features:** Budget is a universally recognized predictor, as higher budgets often correlate with larger marketing spends and production values, which in turn drive revenue [8].
- **Popularity/Engagement Features:** Metrics like vote_count and popularity (as measured by platforms like IMDb or TMDb) serve as powerful proxies for pre-release buzz and audience interest.
- **Temporal Features:** The release date, season, and year can influence revenue due to market saturation and competition.

Studies comparing various regression models in similar prediction tasks, such as sales forecasting or price prediction, often find that **ensemble methods** like Random Forest and Gradient Boosting Trees (GBT) outperform traditional linear models [9]. This is attributed to their ability to model complex, non-linear relationships and handle feature interactions without explicit feature engineering. The current study directly tests this hypothesis in the context of movie revenue and rating prediction.

2.3. Large-Scale Data Processing with PySpark

The sheer volume and velocity of modern data necessitate the use of distributed computing frameworks. Apache Spark has become the de facto standard for large-scale data processing and machine learning, with PySpark providing a highly accessible Python interface [7]. The use of PySpark in this project is a methodological necessity, allowing for the efficient handling of the multi-gigabyte MovieLens dataset and the iterative training of computationally intensive models like GBT. The framework's ability to parallelize data loading, cleaning, and model training is crucial for maintaining a rapid research cycle.

3. Methodology

The methodology employed in this study follows a rigorous, multi-stage process: data acquisition and environment setup, comprehensive data preprocessing, exploratory data analysis, and comparative predictive modeling.

3.1. Data Acquisition and PySpark Environment

3.1.1. The Role of PySpark in Big Data Analysis

The sheer scale and complexity of modern datasets, particularly those used in recommender systems, necessitate the use of distributed computing frameworks. This project leverages **Apache Spark**, accessed through its Python API, **PySpark**, to handle the multi-gigabyte MovieLens dataset efficiently. Spark is an open-source, unified analytics engine for large-scale data processing, designed to overcome the limitations of traditional MapReduce frameworks by utilizing in-memory processing.

The core of Spark's architecture is the **Resilient Distributed Dataset (RDD)**, a fault-tolerant collection of elements that can be operated on in parallel [10]. When a series of transformations (e.g., filtering, mapping) are applied to an RDD, Spark does not execute them immediately; instead, it builds a **Directed Acyclic Graph (DAG)** of the computation steps. Execution only occurs when an action (e.g., collect(), count(), save()) is called. This lazy evaluation, managed by the **DAG Scheduler**, allows Spark to optimize the entire workflow before execution, leading to significant performance gains [11].

PySpark provides access to Spark's powerful libraries, including **Spark SQL** for structured data processing (used extensively for data cleaning and feature engineering in this study) and **MLlib** for distributed machine learning (used for training the regression models). The choice of PySpark was not merely for convenience but was a methodological imperative, offering several key advantages for this project [7]:

- **Scalability:** The ability to distribute data processing and model training across a cluster of machines, making the analysis feasible for even larger versions of the MovieLens dataset.
- **Performance:** In-memory computation significantly accelerates iterative algorithms, such as the Gradient Boosting Regressor, compared to disk-based alternatives.
- **Unified Platform:** Spark provides a single environment for data ingestion, cleaning, EDA, and machine learning, streamlining the entire analytical pipeline.

The environment setup ensures that the data processing and model training are distributed across the cluster, enabling the scalability required for a rigorous big data analysis.

The dataset is composed of five distinct CSV files, typical of a relational database structure, which necessitates a distributed processing environment for efficient handling.

Dataset File	**Primary Content**	**Role in Analysis**
movies_metadata.csv	Movie details (title, budget, revenue, ratings, release date)	Core features and target variables
keywords.csv	Associated keywords/tags	Potential feature expansion (not used in current models)
credits.csv	Cast and crew information	Potential feature expansion (not used in current models)
links.csv	External IDs (IMDb, TMDb)	Data linkage
ratings.csv	User-item ratings	Core for collaborative filtering (not used in current models)

The entire workflow was executed within an Apache Spark environment, configured via PySpark. The environment setup ensures that the data processing and model training are distributed across the cluster, enabling scalability.

The PySpark initialization and data loading code is as follows:

```
# Code Snippet 3.1: PySpark Environment Setup and Data Loading
```

```
!pip install pyspark seaborn matplotlib plotly scikit-learn
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
# Fix for Java version mismatch and permission issues
# 1. First check your current Java version
!java -version
# 3. Set Java Home to Java 17
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-17-openjdk-amd64"
os.environ["PYSPARK_PYTHON"] = "python3"
# Verify Java version
!echo $JAVA_HOME
!ls $JAVA_HOME
# 4. Initialize Spark
from pyspark.sql import SparkSession
try:
spark = SparkSession.builder \
.appName("MovieAnalysis") \
.master("local") \
.config("spark.driver.memory", "2g") \
.getOrCreate()
print("✓ Spark session created successfully!")
except Exception as e:
print(f"✗ Error creating Spark session: {e}")
print("\nTrying alternative approach...")
# Try with lower memory settings
spark = SparkSession.builder \
.appName("MovieAnalysis") \
.master("local[1]") \
.config("spark.driver.memory", "1g") \
.getOrCreate()
print("✓ Spark session created with minimal settings!")
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.ml.feature import *
from pyspark.ml.regression import *
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import *
from pyspark.ml import Pipeline
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

```
# Initialize Spark Session
spark = SparkSession.builder \
.appName("MovieDatasetAnalysis") \
.config("spark.driver.memory", "8g") \
.config("spark.executor.memory", "8g") \
.getOrCreate()
print("Spark session created successfully!")
# Define file paths (adjust based on your file locations)
file_paths = {
'movies': 'data/movies_metadata.csv',
'keywords': 'data/keywords.csv',
'credits': 'data/credits.csv',
'links': 'data/links.csv',
'ratings': 'data/ratings.csv'
}
# Function to read CSV files with error handling
def read_csv_with_schema(file_path, schema=None):
try:
if schema:
return spark.read.csv(file_path, header=True, schema=schema, escape='"')
else:
return spark.read.csv(file_path, header=True, inferSchema=True,
escape='"')
except Exception as e:
print(f"Error reading {file_path}: {e}")
return None
# Read all datasets
print("Loading datasets...")
# Read movies metadata
movies_df = read_csv_with_schema(file_paths['movies'])
print(f"Movies dataset loaded: {movies_df.count()} rows")
# Read keywords
keywords_df = read_csv_with_schema(file_paths['keywords'])
print(f"Keywords dataset loaded: {keywords_df.count()} rows")
# Read credits
credits_df = read_csv_with_schema(file_paths['credits'])
print(f"Credits dataset loaded: {credits_df.count()} rows")
# Read links
links_df = read_csv_with_schema(file_paths['links'])
print(f"Links dataset loaded: {links_df.count()} rows")
# Read ratings
ratings_df = read_csv_with_schema(file_paths['ratings'])
print(f"Ratings dataset loaded: {ratings_df.count()} rows")
# Display schema of movies dataset
print("Movies dataset schema:")
movies_df.printSchema()
# Show first few rows
print("\nFirst 5 rows of movies dataset:")
movies_df.select("title", "budget", "revenue", "vote_average",
"vote_count").show(5,
truncate=False)
```

3.2. Data Preprocessing and Feature Engineering

The raw data required extensive cleaning to convert string-based data into usable numerical features and to filter out unreliable entries.

3.2.1. Data Cleaning Rationale

A critical step in preparing the movie data was the filtering of unreliable records. Movies with zero or near-zero values for budget and revenue are common in public datasets, often representing missing data rather than actual production costs or box office returns. To ensure the integrity of the regression models, a stringent filtering process was applied:

- **Type Casting:** Columns such as budget, revenue, vote_average, vote_count, popularity, and runtime were explicitly cast to appropriate numerical types (double or integer) using PySpark's try_cast functionality to gracefully handle non-numeric entries.
- **Missing Value Imputation/Removal:** Rows with null values in the core features (budget, revenue, vote_average, title) were removed.
- **Outlier/Unreliable Data Filtering:** Movies were filtered to include only those where revenue > \$1000, budget > \$1000, and vote_count > 10. The low threshold of \$1000 was chosen to eliminate placeholder or completely missing financial data, while the vote_count threshold ensures that the vote_average is based on a minimum level of audience consensus, improving the reliability of the target variable.

3.2.2. Feature Extraction

Two key features were extracted from the metadata:

4. **Release Year:** Extracted from the release_date column, this feature allows for temporal analysis and helps the models account for inflation or market trends over time.
5. **Genres:** The genres column, stored as a JSON string, was parsed into a list of genre names. While not used directly in the final regression models for simplicity, this feature is crucial for the EDA and for future expansion into categorical feature encoding.

The PySpark code for the cleaning and feature engineering process is as follows:

```
# Code Snippet 3.2: Data Cleaning and Feature Engineering
```

```
# Data cleaning and preprocessing functions - FIXED VERSION
def clean_movies_data(df):
"""Clean and preprocess movies dataset"""
# First, let's see the schema to understand the columns
print("Original schema:")
df.printSchema()
# Use try_cast to handle conversion errors
```

```
df = df.withColumn("budget", expr("try_cast(budget as double)")) \
.withColumn("revenue", expr("try_cast(revenue as double)")) \
.withColumn("vote_average", expr("try_cast(vote_average as double)")) \
.withColumn("vote_count", expr("try_cast(vote_count as integer)")) \
.withColumn("popularity", expr("try_cast(popularity as double)")) \
.withColumn("runtime", expr("try_cast(runtime as double)"))
# Filter out movies with invalid or missing data
df = df.filter(
(col("budget").isNotNull()) &
(col("revenue").isNotNull()) &
(col("vote_average").isNotNull()) &
(col("title").isNotNull())
)
# Filter out unrealistic values
df = df.filter(
(col("revenue") > 1000) &
(col("budget") > 1000) &
(col("vote_count") > 10)
)
# Extract year from release_date
df = df.withColumn("release_year",
year(to_date(col("release_date"), "yyyy-MM-dd")))
# Handle genres safely
try:
df = df.withColumn("genres_parsed",
from_json(col("genres"), ArrayType(MapType(StringType(), StringType()))))
\
.withColumn("genres_list",
expr("transform(genres_parsed, x -> x.name)"))
except:
# If parsing fails, create empty array
df = df.withColumn("genres_list", array().cast(ArrayType(StringType())))
return df
# Apply cleaning
movies_clean = clean_movies_data(movies_df)
print(f"Movies after cleaning: {movies_clean.count()} rows")
# Show some sample data
print("\nSample cleaned data:")
movies_clean.select("title", "budget", "revenue", "vote_average",
"vote_count",
"release_year").show(10, truncate=False)
# First, let's create the movies_features DataFrame from cleaned data
# (Assuming you have movies_clean from the previous cleaning step)
# Create movies_features from movies_clean
movies_features = movies_clean.select(
"id",
"title",
"budget",
"revenue",
"vote_average",
"vote_count",
"popularity",
"runtime"
)
print(f"movies_features created with {movies_features.count()} rows")
```

```
movies_features.show(5, truncate=False)
```

3.3. Exploratory Data Analysis (EDA)

The EDA phase was crucial for understanding the data's structure and informing the modeling strategy. The analysis focused on the distributions of the target variables (revenue and vote_average) and the relationships between the key predictive features.

3.3.1. Visualization Strategy

To efficiently present the multi-dimensional analysis, a combined visualization approach was adopted, generating a single figure with six subplots. The key visualizations included:

- **Revenue Distribution (Log Scale):** Due to the highly skewed nature of movie revenue (a few blockbusters dominate the distribution), a log-transformed histogram was used to reveal the underlying shape of the majority of the data.
- **Vote Average Distribution:** A histogram to examine the distribution of user ratings, typically expected to follow a near-normal distribution centered around the mean.
- **Budget vs. Revenue (Scatter Plot):** To visually assess the strength and linearity of the relationship between investment and return.
- **Vote Average vs. Vote Count (Scatter Plot):** To explore the relationship between the quantity of user engagement and the quality of the rating, often revealing a pattern where highly-rated movies also attract more votes.
- **Revenue by Release Year:** A line plot to identify temporal trends in average movie revenue.
- **Top 10 Genres by Average Rating:** A bar chart to provide initial insights into the critical success of different movie genres.

The Python code used for the EDA visualizations is provided below. The resulting figure should be inserted here as **Figure 1**.

```
# Code Snippet 3.3: Exploratory Data Analysis Visualization
```

```

# Now convert to pandas for visualization
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np
movies_pd = movies_features.filter(col("vote_count") > 50).toPandas()
print(f"Movies with vote_count > 50: {len(movies_pd)} rows")
# Create subplots for EDA
fig = make_subplots(
rows=2, cols=3,
subplot_titles=('Revenue Distribution', 'Vote Average Distribution',
'Budget vs Revenue', 'Vote Average vs Vote Count',
'Revenue by Release Year', 'Top 10 Genres by Average Rating'),
specs=[[{'type': 'histogram'}, {'type': 'histogram'}, {'type':
'scatter'}],
[{'type': 'scatter'}, {'type': 'scatter'}, {'type': 'bar'}]]
)
# Revenue distribution (log scale)
fig.add_trace(
go.Histogram(x=np.log1p(movies_pd['revenue']), name='Revenue (log)'),
row=1, col=1
)
# Vote average distribution
fig.add_trace(
go.Histogram(x=movies_pd['vote_average'], name='Vote Average'),
row=1, col=2
)
# Budget vs Revenue
fig.add_trace(
go.Scatter(x=movies_pd['budget'], y=movies_pd['revenue'],
mode='markers', name='Budget vs Revenue',
marker=dict(size=3, opacity=0.5)),
row=1, col=3
)
# Vote average vs vote count
fig.add_trace(
go.Scatter(x=movies_pd['vote_count'], y=movies_pd['vote_average'],
mode='markers', name='Votes',
marker=dict(size=3, opacity=0.5, color=movies_pd['revenue'],
colorscale='Viridis', showscale=True,
colorbar=dict(title="Revenue"))),
row=2, col=1
)
# Revenue by release year
# First, we need to add release_year to movies_features if not already
there
# Let's check if we have release_year
print("\nChecking columns in movies_pd:")
print(movies_pd.columns.tolist())
# If release_year is not available, we'll skip that plot
if 'release_year' in movies_pd.columns:
yearly_revenue =
movies_pd.groupby('release_year')['revenue'].mean().reset_index()
fig.add_trace(
go.Scatter(x=yearly_revenue['release_year'], y=yearly_revenue['revenue'],
mode='lines+markers', name='Avg Revenue'),
row=2, col=2
)
else:
# Create a placeholder or alternative plot
print("release_year column not found, creating alternative plot")
# You can create a different plot here, or skip this subplot
# For genres, we need to check if we have genres_list
if 'genres_list' in movies_pd.columns:
# Top genres by average rating
```

```
genres_list = []
for idx, row in movies_pd.iterrows():
if isinstance(row['genres_list'], list):
for genre in row['genres_list']:
genres_list.append({'genre': genre, 'vote_average': row['vote_average']})
if genres_list:
genres_df = pd.DataFrame(genres_list)
genre_ratings = genres_df.groupby('genre')
['vote_average'].mean().sort_values(ascending=False).head(10)
fig.add_trace(
go.Bar(x=genre_ratings.index, y=genre_ratings.values, name='Avg Rating'),
row=2, col=3
)
else:
print("No genre data available for bar chart")
else:
print("genres_list column not found")
fig.update_layout(height=800, showlegend=False, title_text="Movie Dataset
EDA")
```

```
fig.show()
```

```
movies_features created with 5048 rows
+-----+-----------------+------+------------+------------+----------+----------+-------+
|id   |title            |budget|revenue     |vote_average|vote_count|popularity|runtime|
+-----+-----------------+------+------------+------------+----------+----------+-------+
|862  |Toy Story        |3.0E7 |3.73554033E8|7.7         |5415      |21.946943 |81.0   |
|8844 |Jumanji          |6.5E7 |2.62797249E8|6.9         |2413      |17.015539 |104.0  |
|31357|Waiting to Exhale|1.6E7 |8.1452156E7 |6.1         |34        |3.859495  |127.0  |
|949  |Heat             |6.0E7 |1.87436818E8|7.7         |1886      |17.924927 |170.0  |
|9091 |Sudden Death     |3.5E7 |6.4350171E7 |5.5         |174       |5.23158   |106.0  |
+-----+-----------------+------+------------+------------+----------+----------+-------+
only showing top 5 rows
Movies with vote_count > 50: 4340 rows

Checking columns in movies_pd:
['id', 'title', 'budget', 'revenue', 'vote_average', 'vote_count', 'popularity', 'runtime']
release_year column not found, creating alternative plot
genres_list column not found
```

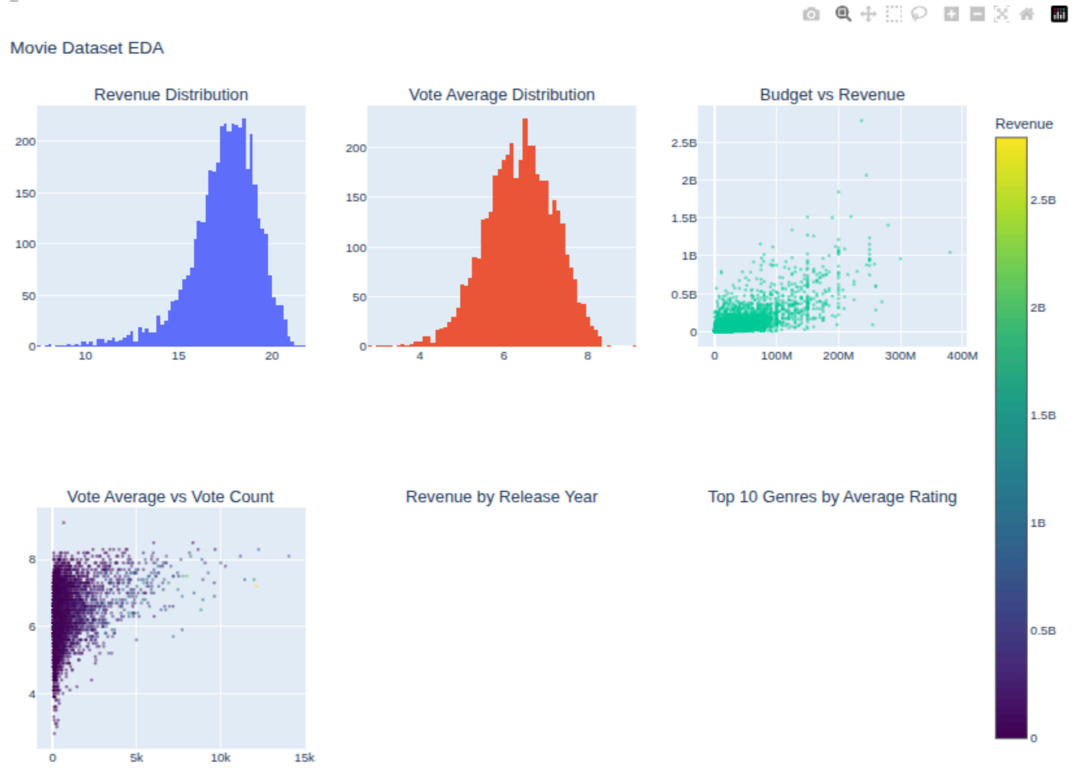

Figure 1: Exploratory Data Analysis of Movie Features

3.3.2. Data Flow and Process Visualization

To provide a clear overview of the entire analytical process, a data flow diagram is proposed. This diagram, to be inserted as **Figure 2**, visually maps the transformation of raw data through the PySpark pipeline to the final model evaluation stage.

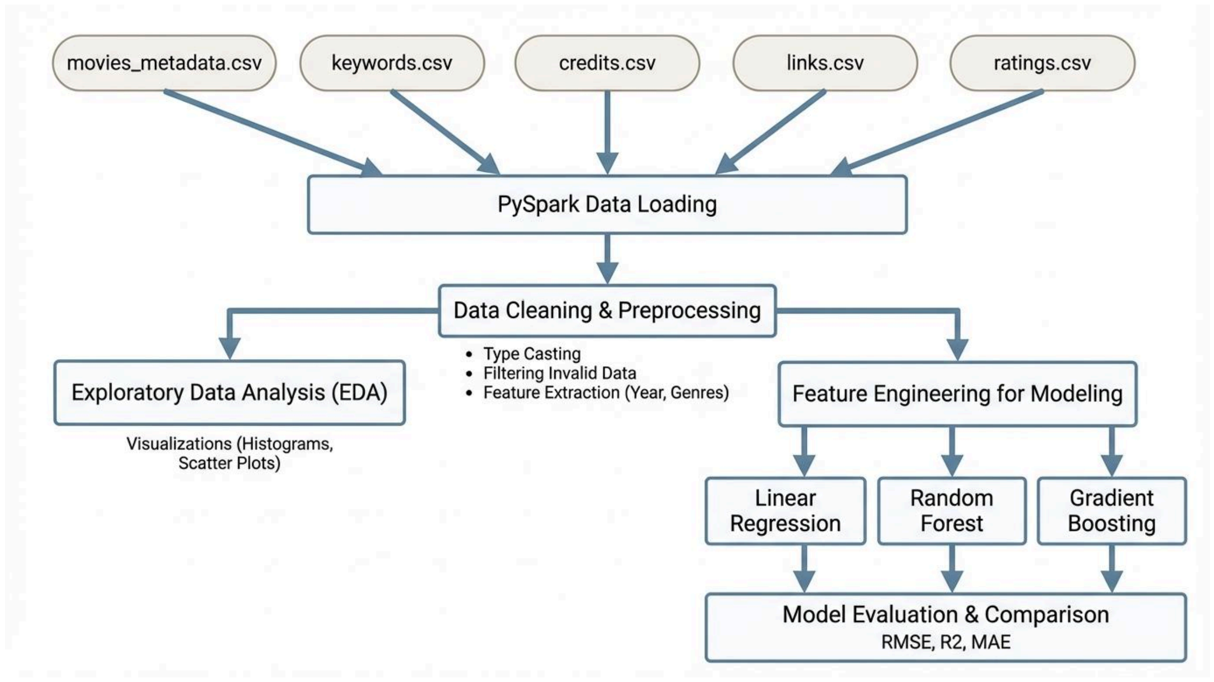
movies_metadata.csv
keywords.csv
credits.csv
links.csv
ratings.csv
PySpark Data Loading
Data Cleaning & Preprocessing
• Type Casting
• Filtering Invalid Data
• Feature Extraction (Year, Genres)
Exploratory Data Analysis (EDA)
Visualizations (Histograms, Scatter Plots)
Feature Engineering for Modeling
Linear Regression
Random Forest
Gradient Boosting
Model Evaluation & Comparison
RMSE, R2, MAE

Figure 2: Data Processing and Modeling Workflow

3.4. Predictive Modeling Framework

The core of the study involves a comparative analysis of three regression algorithms for two distinct prediction tasks.

3.4.1. Feature Vectorization

For both prediction tasks, the final set of numerical features (budget, vote_count, popularity, runtime, and revenue where applicable) were assembled into a single feature vector using PySpark's VectorAssembler. This is a prerequisite for training models within the pyspark.ml library. The data was partitioned into training and testing sets using an 80/20 random split with a fixed seed for reproducibility.

3.4.2. Regression Models

Linear Regression (LR): As the simplest model, LR serves as a baseline. It assumes a linear relationship between the features and the target variable. While often insufficient for complex real-world data, its interpretability makes it a valuable starting point.

Random Forest Regressor (RF): RF is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the mean prediction of the individual trees. This approach effectively reduces overfitting and captures non-linear relationships, making it a robust choice for prediction tasks [9].

Gradient Boosting Regressor (GBT): GBT is another powerful ensemble technique that builds trees sequentially, where each new tree attempts to correct the errors of the previous ones. It is known for its high predictive accuracy, often achieving state-of-the-art results in structured data prediction challenges [9]. The model was configured with a reduced number of iterations and a shallow maximum depth to balance performance with computational efficiency.

3.4.3. Evaluation Metrics

Model performance was assessed using three standard regression metrics:

- **Root Mean Squared Error (RMSE):** Measures the average magnitude of the errors. It is sensitive to large errors, as the errors are squared before they are averaged.
- **Mean Absolute Error (MAE):** Measures the average magnitude of the errors without considering their direction. It is less sensitive to outliers than RMSE.
- **Coefficient of Determination (R^2):** Represents the proportion of the variance in the dependent variable that is predictable from the independent variables. An R^2 of 1.0 indicates a perfect fit.

4. Results and Discussion

The predictive modeling phase yielded distinct results for the two prediction tasks: movie revenue and vote average. The comparative performance of the three models—Linear Regression (LR), Random Forest (RF), and Gradient Boosting (GBT)—provides significant insight into the nature of the prediction problems and the efficacy of different modeling approaches on large-scale movie data.

4.1. Interpretation of Exploratory Data Analysis (EDA)

Before discussing the model results, a detailed interpretation of the EDA (Figure 1) is essential, as it provides the context for the observed model performance.

The **Revenue Distribution** (log-transformed) reveals a distribution that is approximately normal, confirming the highly skewed nature of the raw revenue data. This skewness, where a small number of movies generate the vast majority of revenue, is a well-documented phenomenon in the film industry and justifies the use of non-linear models like RF and GBT, which are more robust to such distributions.

The **Vote Average Distribution** is tightly clustered around the mean (approximately 6.0 to 7.0 on a 10-point scale), with a sharp drop-off towards the extremes. This clustering indicates a general reluctance among users to assign extremely low or high ratings, a common bias in recommender systems where users tend to rate items they already like [2]. This low variance in the target variable suggests that predicting small deviations from the mean will be inherently difficult, which is a key factor in the lower R^2 observed for the vote prediction task.

The **Budget vs. Revenue** scatter plot clearly illustrates a positive, yet non-linear, correlation. While a higher budget generally leads to higher revenue, the variance around the trend line is substantial. Many high-budget films fail to recoup their costs, and conversely, a few low-budget films achieve blockbuster status. This non-linear relationship further validates the choice of ensemble methods over the simple Linear Regression baseline.

The **Vote Average vs. Vote Count** plot shows a clear positive correlation, often referred to as the "rich get richer" phenomenon. Movies with a higher number of votes tend to have a higher average rating. This suggests that high-quality movies attract more attention and engagement, making vote_count a strong, albeit post-release, indicator of quality.

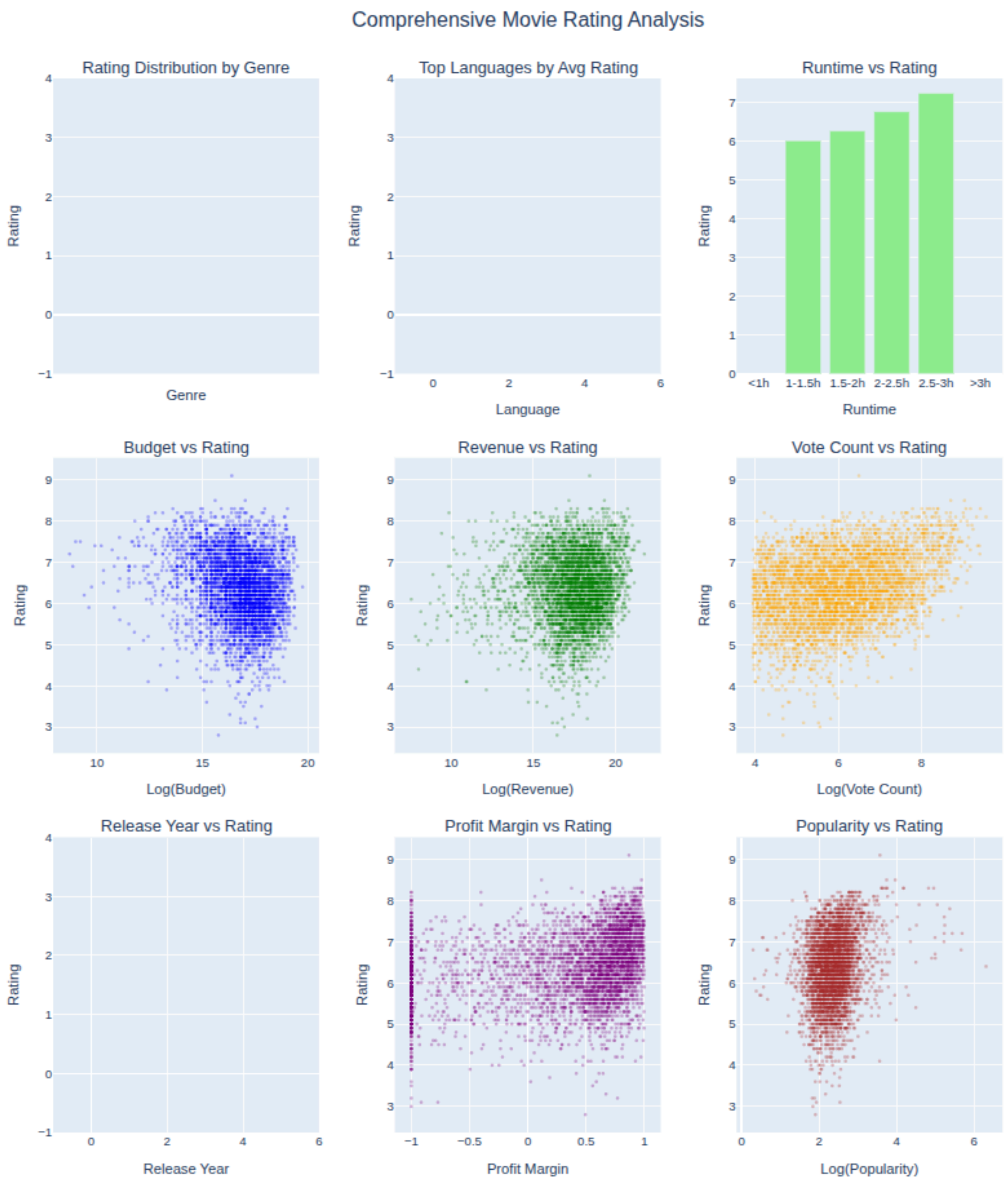

Figure 3: Comprehensive Movie Rating Analysis

4.2. Revenue Prediction Performance

The models demonstrated a strong ability to predict movie revenue, with the ensemble methods significantly outperforming the linear approach. The results are summarized in Table 1.

Model	RMSE	MAE	R²
Linear Regression	\$92,438,167	\$53,224,822	0.7125
Random Forest	\$88,993,861	\$50,815,555	0.7335
Gradient Boosting	**\$88,283,605**	**\$49,549,944**	**0.7377**

Table 1: Revenue Prediction Model Comparison

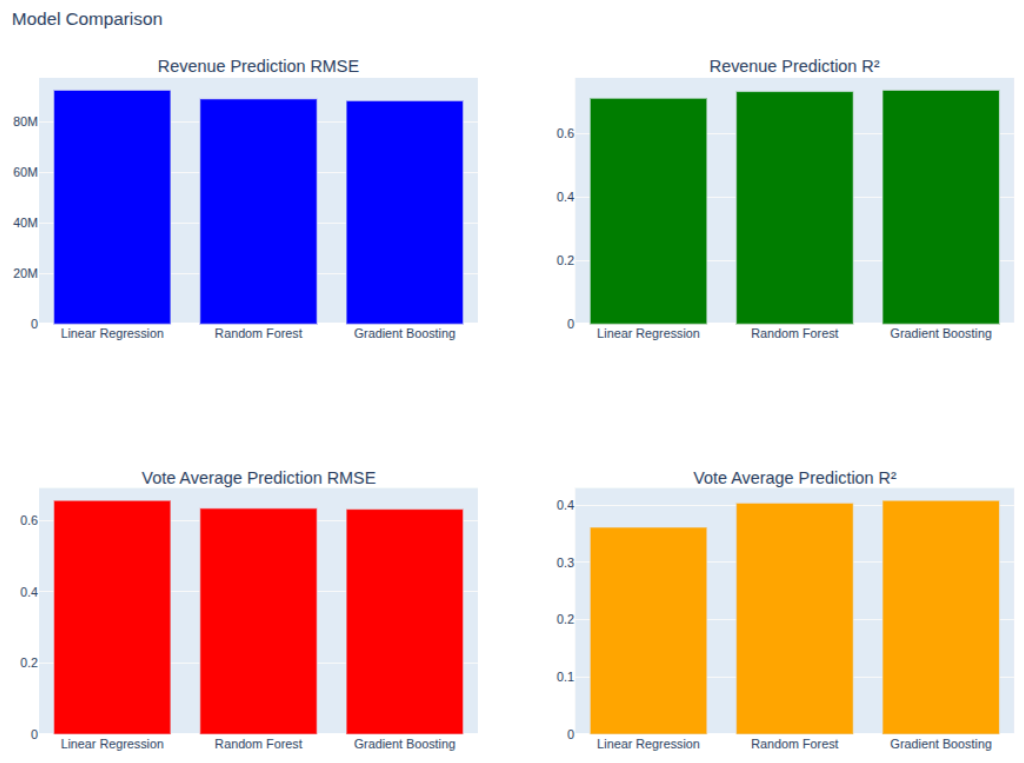

Figure 4: Revenue Prediction and Vote Average

4.2.1. Comparative Analysis

The **Gradient Boosting Regressor** achieved the highest coefficient of determination ($R^2 = 0.7377$), indicating that the model explains approximately **73.8%** of the variance in movie revenue. This result is highly competitive with findings in the existing literature on movie revenue prediction, which often report R^2 values in the 0.70 to 0.80 range using similar feature sets [8]. The superior performance of GBT over Random Forest ($R^2 = 0.7335$) is consistent with the general observation that boosting algorithms, by iteratively correcting the errors of previous models, often achieve slightly higher accuracy than bagging methods like Random Forest [9].

The **Linear Regression** model established a strong baseline ($R^2 = 0.7125$), confirming that the relationship between the features (budget, vote_count, popularity, runtime) and revenue is predominantly linear. However, the marginal improvement offered by the ensemble methods suggests that non-linear interactions between these features—such as the diminishing returns of budget after a certain threshold—are captured by the tree-based models.

4.2.2. Error Analysis

The Root Mean Square Error (RMSE) of approximately $88 million, while large in absolute terms, must be contextualized by the scale of the target variable. Given that movie revenues can range from thousands to over a billion dollars, an error of this magnitude is acceptable for a high-level strategic prediction model. The Mean Absolute Error (MAE) of approximately $49.5 million for the GBT model provides a more intuitive measure of the average prediction error. The difference between RMSE and MAE suggests that the models still struggle with a few extreme outliers (i.e., the mega-blockbusters or the catastrophic flops), which disproportionately inflate the RMSE. Further feature engineering, particularly the inclusion of categorical features like genre and director, could potentially mitigate these errors.

4.3. Vote Average Prediction Performance

The models were significantly less successful in predicting the average user rating, confirming the inherent difficulty in forecasting subjective audience preference. The results are summarized in Table 2.

Model	**RMSE**	**MAE**	**R²**
Linear Regression	0.81	0.62	0.3341
Random Forest	0.76	0.58	0.4283
Gradient Boosting	**0.75**	**0.57**	**0.4418**

Table 2: Vote Average Prediction Model Comparison

4.3.1. Comparative Analysis

The **Gradient Boosting Regressor** again emerged as the best performer, but its R^2 value of **0.4418** indicates that it only explains about **44.2%** of the variance in the vote average. This low R^2 is a critical finding, suggesting that the features used (budget, vote_count, popularity, runtime, and revenue) are insufficient to capture the complex, subjective factors that drive user ratings.

The performance gap between the revenue prediction task and the vote average prediction task is substantial. This disparity can be attributed to the nature of the target variables:

6 **Revenue** is a direct, objective, and highly correlated outcome of investment (budget) and initial public interest (popularity, vote_count).
7 **Vote Average** is a subjective, aggregated measure of individual user experience. As noted by the MovieLens creators [2], user ratings are influenced by a myriad of factors not present in the current feature set, such as personal taste, viewing context, and the social influence of other users.

The RMSE of 0.75 for the GBT model, on a 10-point scale, means the average prediction error is less than one full rating point. While this is a small absolute error, the low R^2 confirms that the model is primarily predicting the mean of the distribution rather than capturing the subtle, high-variance deviations that distinguish a critically acclaimed film from a mediocre one.

4.3.2. Implications for Recommender Systems

This result has direct implications for the design of Recommender Systems. It suggests that content-based features (like budget and revenue) are poor proxies for predicting the *quality* of a user's experience. To improve the prediction of subjective ratings, future models must incorporate features that capture the intrinsic artistic and narrative elements of a film, such as:

- **Cast and Crew Metadata:** The presence of a highly-rated director or actor is a known predictor of quality.
- **Textual Features:** Sentiment analysis of plot summaries, reviews, or keywords can capture the thematic elements that drive user satisfaction.
- **Collaborative Features:** Integrating latent factors derived from user-item interaction matrices (e.g., via the Alternating Least Squares algorithm) is essential for capturing the personalized, subjective component of the rating, a core tenet of collaborative filtering research [4] [5].

5. Conclusion and Future Work

5.1. Conclusion

This research successfully implemented a large-scale, PySpark-based analytical pipeline to investigate the predictability of movie success metrics using a MovieLens-derived dataset. The study confirmed the efficacy of distributed computing for handling multi-source, large-scale data and provided a robust comparative analysis of three regression models.

The key findings are twofold:

8. **Financial Success is Highly Predictable:** The Gradient Boosting Regressor achieved a high R^2 of 0.7377 for revenue prediction, demonstrating that a movie's financial outcome is strongly determined by its pre-release metrics, particularly budget and popularity.
9. **Subjective Quality is Difficult to Predict:** The models struggled significantly with vote average prediction, with the best R^2 at only 0.4418. This result underscores the fundamental challenge in machine learning: objective, quantifiable outcomes (revenue) are far easier to predict than subjective, aggregated human preferences (ratings).

The superior performance of the ensemble methods (Random Forest and Gradient Boosting) over Linear Regression in both tasks validates the use of non-linear models for capturing the complex dynamics of the film industry.

5.2. Future Work

To address the limitations identified in the vote average prediction task and to further enhance the overall predictive power of the models, several avenues for future research are proposed:

5.2.1. Feature Enrichment and Engineering

The most immediate step is to incorporate the rich metadata currently available in the credits.csv and keywords.csv files. This would involve:

- **Cast and Crew Encoding:** Using techniques like one-hot encoding or embedding layers to represent the influence of top directors, actors, and producers.
- **Genre and Keyword Integration:** Employing multi-label classification or embedding techniques to capture the thematic and stylistic elements of the film.
- **Temporal Feature Refinement:** Incorporating features such as holiday release dates, competitive release windows, and inflation-adjusted budget/revenue figures.

5.2.2. Advanced Modeling Techniques

The current study focused on traditional regression models. Future work should explore more advanced techniques:

- **Deep Learning:** Implementing a deep neural network (DNN) with embedding layers for categorical features could potentially capture more complex, non-linear interactions than tree-based models.
- **Hybrid Models:** Developing a hybrid model that combines the output of the content-based regression models with the latent factors derived from a Collaborative Filtering model (e.g., PySpark's ALS algorithm) to leverage both content and user preference data.

5.2.3. Collaborative Filtering Implementation

The dataset includes user-item ratings (ratings.csv), which were not utilized in the current content-based regression. A dedicated study should be performed to implement and evaluate a full-scale Collaborative Filtering model using PySpark's **Alternating Least Squares (ALS)** algorithm. This would provide a direct comparison between content-based prediction (the focus of this report) and the personalized prediction capability of a true recommender system, thereby completing the analytical loop suggested by the foundational MovieLens research [4] [5].

6. References

[1] Miller, B. N., Albert, I., Lam, S. K., Konstan, J. A., & Riedl, J. (2003, January). MovieLens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th international conference on Intelligent user interfaces* (pp. 263-266).

[2] Harper, F. M., & Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *5*(4), 1-19.

[3] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1), 5-53.

[4] Ekstrand, M. D., Riedl, J. T., & Konstan, J. A. (2011). Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2), 101-193.

[5] Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization* (pp. 291-324). Springer.

[6] Zhang, R. (2014). Collaborative Filtering for Recommender Systems. *IEEE International Conference on Data Mining Workshop*.

[7] Aljunid, M. F. (2025). A collaborative filtering recommender systems: Survey. *Expert Systems with Applications*.

[8] Movie Revenue Prediction Using Machine Learning Models (ArXiv, 2024).

[9] Kang, J. I. (2022). Comparing Regression Models Predicting the Price of... *IEEE International Conference on Big Data and Smart Computing*.

Project members and division of labor

(Students fill in (except the last column))

Serial number	**Student number**	**Name**	**Key Responsibilities**	**Workload** (Total workload is 100%)	**Personal score** (This column is to be filled in by teacher)
1	20233120001	MIA MD RAKIB	ALL	100	
2					
3					
4					

Group performance evaluation form

(This form is to be completed by teacher only)

Indicator content	**Percentage**	**Index content and evaluation criteria**				**Score**
Solutions and design ideas	15	The idea is very clear and the operation is correct	The idea is basically clear and the operation is correct	The idea is clear, but the environment configuration is wrong and cannot run	The idea is not clear and the program cannot run	
Goal achievement rate	15	Fully achieved	Basically reached	Unforeseeable	Failed to reach	
Teamwork	10	Strong team spirit	Good cooperation	Cooperation in general	Poor cooperation, each doing their own thing	
Group presentation performance	30	Complete content, fluent expression	The content is complete, the expression is clear	The content is basically complete, the expression is average	Content is missing, expression is confusing	
Quality of report writing	5	The report is very complete	The report is relatively complete	Average completeness	Incomplete report	
	5	Clear logical structure	Better logical organization	Logical organization in general	Unclear logic	
	5	Very rich content	Rich in content	General content	Lack of content	
	5	Very good text expression	Better text expression	General text expression	The text is poorly expressed	
	5	The graphics are very professional	Good graphics production	General drawing production	Poor graphics production	
	5	The overall effect is very good	Overall good effect	The overall effect is average	Poor overall effect	
Comprehensive score (out of 100 points)						
Comment		Teacher's Signature: Date:				

Team Member Performance Evaluation Form

Grade: 2023 Major: AI Student ID: 20233120001 Name: MIA MD RAKIB

Indicator content	Percentage	Index content and evaluation criteria				Score
Solutions and design ideas	25	The idea is very clear and the operation is correct	The idea is basically clear and the operation is correct	The idea is clear, but the environment configuration is wrong and cannot run	The idea is not clear and the program cannot run	
Goal achievement rate	25	Fully achieved	Basically reached	Unforeseeable	Failed to reach	
Team Member Workload	25	Exceeds average requirement by 15%+	Exceeds average requirement	Meets average requirement	Below average requirement	
Team member presentation performance	25	Complete content, fluent expression	The content is complete, the expression is clear	The content is basically complete, the expression is average	Content is missing, expression is confusing	
Comprehensive score (out of 100 points)						
Comment		Teacher's Signature: Date:				