

Configuration Manual

MSc Research Project
MSc AI

Shubham Kawade
Student ID: X23354658

School of Computing
National College of Ireland

Supervisor: Prof. Sheresh Zahoor

National College of Ireland
MSc Project Submission Sheet



School of Computing

Shubham Pandurang Kawade

Student Name:

Student ID: ...x23354658.....

Programme:MSc. Artificial Intelligence.....
Year:2024-2025.....

Module:MSc. Research Project.....

Lecturer:Prof Sheresh Zahoor.....

Submission Due Date:01/09/2025.....

Project Title:Predicting Box Office Success with Advanced Regression Techniques: Using Data to Merge Art and Analytics.....

Word Count: **Page Count:** ...14.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

SignatureShubham
: kawade.....

Date:01/09/2025.....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Forename Surname

Student ID:

1 Introduction

This Configuration Manual provides detailed instructions to replicate the computational environment and workflow for the MSc research project, *Predicting Box Office Success with Advanced Regression Techniques: Using Data to Merge Art and Analytics*. The project predicts box office revenue and IMDb ratings using the Kaggle IMDb Movie Dataset (1,000 films, CC-BY license) with regression (Linear, Ridge, Lasso) and tree-based models (Random Forest, Gradient Boosting, XGBoost). The manual covers system requirements, environment setup, data acquisition and preprocessing, model implementation, output generation, and troubleshooting, ensuring reproducibility for researchers and practitioners. It aligns with the thesis's Methodology (Section 3), Design Specification (Section 4), and Implementation (Section 5) chapters, using Python 3.8+ and libraries like scikit-learn and xgboost.

2 System Requirements

2.1 Hardware Requirements

- **Processor:** Intel i5/i7 or equivalent (used: Intel i7).
- **Memory:** Minimum 8GB RAM, recommended 16GB (used: 16GB).
- **Storage:** 5GB free disk space for dataset, code, and outputs.
- **Operating System:** Windows 10/11, macOS, or Linux (used: Windows/Linux).

2.2 Software Requirements

- **Python:** Version 3.8 or higher.
- **Libraries:**
 - ✓ pandas (1.5.3): Data manipulation.
 - ✓ numpy (1.24.3): Numerical computations.
 - ✓ scikit-learn (1.2.2): Regression and tree-based models, preprocessing, evaluation.
 - ✓ xgboost (2.0.3): XGBoost model implementation.
 - ✓ seaborn (0.12.2), matplotlib (3.7.1): Visualizations.
 - ✓ scipy (1.10.1): Statistical analysis.
- **Package Manager:** pip or conda for dependency installation.
- **IDE:** Jupyter Notebook, VS Code, or PyCharm (used: Jupyter Notebook).
- **Dataset:** Kaggle IMDb Movie Dataset (imdb_movie_dataset.csv), available at [Kaggle URL] under CC-BY license.

3 Environment Setup

3.1 Installing Python

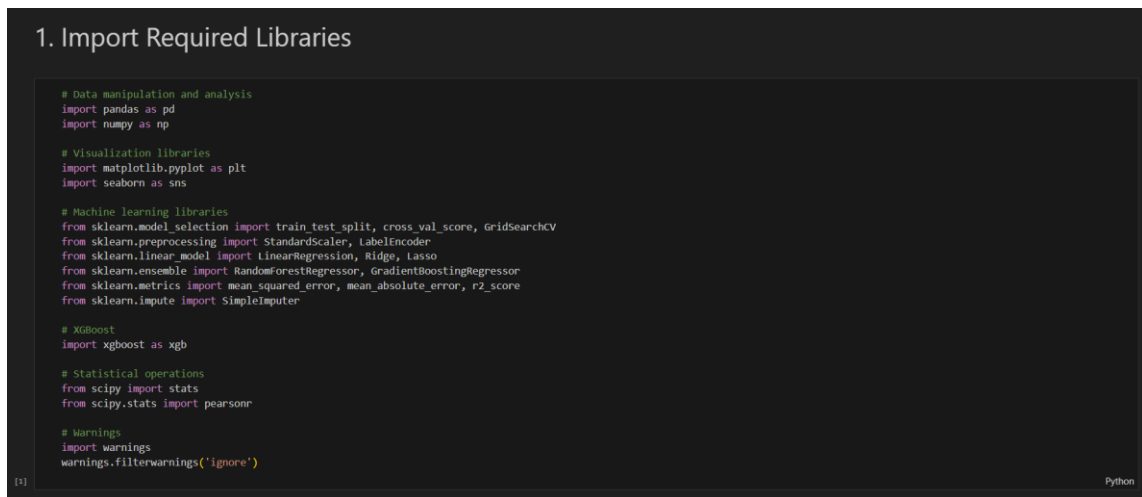
1. Download Python 3.8+ from python.org.
2. Install, ensuring pip is included and added to PATH.
3. Verify installation: `python --version` (should return 3.8 or higher).

3.2 Installing Dependencies

1. Install required libraries using pip:

```
pip install pandas==1.5.3 numpy==1.24.3 scikit-learn==1.2.2 xgboost==2.0.3  
seaborn==0.12.2 matplotlib==3.7.1 scipy==1.10.1
```

2. Verify installations: `pip list` (should list all packages with correct versions).



```
1. Import Required Libraries  
  
# Data manipulation and analysis  
import pandas as pd  
import numpy as np  
  
# Visualization libraries  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Machine learning libraries  
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV  
from sklearn.preprocessing import StandardScaler, LabelEncoder  
from sklearn.linear_model import LinearRegression, Ridge, Lasso  
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor  
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score  
from sklearn.impute import SimpleImputer  
  
# XGBoost  
import xgboost as xgb  
  
# Statistical operations  
from scipy import stats  
from scipy.stats import pearsonr  
  
# Warnings  
import warnings  
warnings.filterwarnings('ignore')
```

Figure 1: Importing Necessary Libraries

4 Data Acquisition and Preprocessing

4.1 Acquiring the Dataset

1. Download the Kaggle IMDb Movie Dataset from [Kaggle URL] (CC-BY license).
2. Save as `imdb_movie_dataset.csv` in the project directory (e.g., `thesis_data/`).
3. Verify file integrity: 1,000 rows, 12 columns (Rank, Title, Genre, Description, Director, Actors, Year, Runtime, Rating, Votes, Revenue, Metascore).

```

# Load the dataset
imdb = pd.read_csv('imdb_movie_dataset.csv')

# Display the first few rows of the dataset
print("Dataset loaded successfully!")
print(imdb.head())

```

Dataset loaded successfully!

Rank	Title	Genre
0	Guardians of the Galaxy	Action,Adventure,Sci-Fi
1	Prometheus	Adventure,Mystery,Sci-Fi
2	Split	Horror,Thriller
3	Sing	Animation,Comedy,Family
4	Suicide Squad	Action,Adventure,Fantasy

Director

Director
James Gunn
Ridley Scott
M. Night Shyamalan
Christophe Yauzelet
David Ayer

Actors

Actors
Chris Pratt, Vin Diesel, Bradley Cooper, Zoe Saldana
Noomi Rapace, Logan Marshall-Green, Michael Fassbender, Charlize Theron

Dataset Info:

```

class 'pandas.core.frame.DataFrame'
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  -
 0   Rank                1000 non-null  int64
 1   Title               1000 non-null  object
 2   Genre               1000 non-null  object
 3   Description         1000 non-null  object
 4   Director            1000 non-null  object
 5   Actors              1000 non-null  object
 6   Year                1000 non-null  int64
 7   Runtime (Minutes)   1000 non-null  int64
 8   Rating              1000 non-null  float64
 9   Votes               1000 non-null  int64
10  Revenue (Millions)  872 non-null   float64
11  Metascore           936 non-null   float64
dtypes: float64(3), int64(4), object(5)
memory usage: 93.9+ KB
None

```

Figure 2: Data Loading and exploration

4.2 Preprocessing Steps

```

print("Dataset Shape:", imdb.shape)

```

Dataset Shape: (1000, 12)

```

print("\nDataset Info:")
print(imdb.info())

```

Dataset Info:

```

class 'pandas.core.frame.DataFrame'
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  -
 0   Rank                1000 non-null  int64
 1   Title               1000 non-null  object
 2   Genre               1000 non-null  object
 3   Description         1000 non-null  object
 4   Director            1000 non-null  object
 5   Actors              1000 non-null  object
 6   Year                1000 non-null  int64
 7   Runtime (Minutes)   1000 non-null  int64
 8   Rating              1000 non-null  float64
 9   Votes               1000 non-null  int64
10  Revenue (Millions)  872 non-null   float64
11  Metascore           936 non-null   float64
dtypes: float64(3), int64(4), object(5)
memory usage: 93.9+ KB
None

```

Figure 3: Data information

```

# Check for missing values
missing_values = imdb.isnull().sum()
missing_percentage = (missing_values / len(imdb)) * 100

missing_df = pd.DataFrame({
    'Missing Count': missing_values,
    'Missing Percentage': missing_percentage
})

print("Missing Values Analysis:")
print(missing_df[missing_df['Missing Count'] > 0])

```

Missing Values Analysis:

	Missing Count	Missing Percentage
Revenue (Millions)	128	12.8
Metascore	64	6.4

```

# Basic statistics
print("\nBasic Statistics:")
imdb.describe()

```

Figure 4: Missing Values and Summary statistics

3. Data Preprocessing and Feature Engineering

```

# Create a copy for preprocessing
df_processed = imdb.copy()

# Handle missing values in Revenue (target variable)
revenue_missing_before = df_processed['Revenue (Millions)'].isnull().sum()
print(f'Revenue missing values before imputation: {revenue_missing_before}')

# Use median imputation for Revenue
revenue_median = df_processed['Revenue (Millions)'].median()
df_processed['Revenue (Millions)'].fillna(revenue_median, inplace=True)

# Handle missing values in Metascore
metascore_missing_before = df_processed['Metascore'].isnull().sum()
print(f'Metascore missing values before imputation: {metascore_missing_before}')

# Use median imputation for Metascore
metascore_median = df_processed['Metascore'].median()
df_processed['Metascore'].fillna(metascore_median, inplace=True)

```

Revenue missing values before imputation: 128

Metascore missing values before imputation: 64

Figure 5: Data Preprocessing and Feature Engineering

```

# Check missing values after imputation
print(f'Revenue missing values after imputation: {df_processed["Revenue (Millions)"].isnull().sum()}')
print(f'Metascore missing values after imputation: {df_processed["Metascore"].isnull().sum()}')

# Handle outliers using IQR method for key numerical features
numerical_features = ['Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)', 'Metascore']

for feature in numerical_features:
    q1 = df_processed[feature].quantile(0.01)
    q3 = df_processed[feature].quantile(0.99)

    # Clip outliers
    df_processed[feature] = df_processed[feature].clip(lower=q1, upper=q3)

    print(f'{feature}: Clipped to range [{q1:.2f}, {q3:.2f}]')

# Apply log transformation to skewed features
# Check skewness first
skewness_before = df_processed[['Revenue (Millions)', 'Votes']].skew()
print("Skewness before transformation:")
print(skewness_before)

```

Revenue missing values after imputation: 0
Metascore missing values after imputation: 0

Runtime (Minutes): Clipped to range [81.99, 166.03]
Rating: Clipped to range [3.90, 8.50]
Votes: Clipped to range [221.08, 865614.18]
Revenue (Millions): Clipped to range [0.03, 423.05]
Metascore: Clipped to range [22.00, 94.01]

Skewness before transformation:
Revenue (Millions) 1.957379
Votes 1.609564
dtype: float64

Figure 6: Handling Missing Values and Outliers

```

# Apply log transformation to skewed features
# Check skewness first
skewness_before = df_processed[['Revenue (Millions)', 'Votes']].skew()
print("Skewness before transformation:")
print(skewness_before)

# Apply log transformation (add 1 to handle zeros)
df_processed['log_revenue'] = np.log1p(df_processed['Revenue (Millions)'])
df_processed['log_votes'] = np.log1p(df_processed['Votes'])

# Check skewness after transformation
skewness_after = df_processed[['log_revenue', 'log_votes']].skew()
print("Skewness after log transformation:")
print(skewness_after)

```

Skewness before transformation:
Revenue (Millions) 1.957379
Votes 1.609564
dtype: float64

Skewness after log transformation:
log_revenue -0.826060
log_votes -1.308716
dtype: float64

Figure 7: Apply Log Transformation

```

# Genre feature engineering - One-hot encoding for multi-label genres
print("Unique genres sample:")
print(df_processed['Genre'].head(10).tolist())

[21] Python

Unique genres sample:
['Action,Adventure,Sci-Fi', 'Adventure,Mystery,Sci-Fi', 'Horror,Thriller', 'Animation,Comedy,Family', 'Action,Adventure,Fantasy', 'Action,Adventure,Fantasy', 'Comedy,Drama,Music', 'Com

# Split genres and create binary features
all_genres = set()
for genres in df_processed['Genre'].dropna():
    genre_list = [g.strip() for g in genres.split(',')]
    all_genres.update(genre_list)

print(f"\nTotal unique genres found: {len(all_genres)}")
print("All genres:", sorted(all_genres))

[22] Python

Total unique genres found: 20
All genres: ['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy', 'History', 'Horror', 'Music', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',

# Create binary features for each genre
for genre in sorted(all_genres):
    df_processed[f'genre_{genre}'] = df_processed['Genre'].str.contains(genre, na=False).astype(int)

print(f"\nGenre features created: {len(all_genres)} binary features")

[23] Python

```

Figure 8: Generic Feature Engineering

```

# Create additional features
# Runtime categories
df_processed['runtime_category'] = pd.cut(df_processed['Runtime (Minutes)'],
                                         bins=[0, 90, 120, 150, float('inf')],
                                         labels=['Short', 'Medium', 'Long', 'Very Long'])

[24] Python

# Rating categories
df_processed['rating_category'] = pd.cut(df_processed['Rating'],
                                         bins=[0, 6.0, 7.0, 8.0, 10.0],
                                         labels=['Low', 'Medium', 'High', 'Excellent'])

[25] Python

# Year-based features
df_processed['decade'] = (df_processed['Year'] // 10) * 10
df_processed['is_recent'] = (df_processed['Year'] >= 2010).astype(int)

[26] Python

# Interaction features
df_processed['rating_votes_interaction'] = df_processed['Rating'] * df_processed['log_votes']
df_processed['runtime_rating_interaction'] = df_processed['Runtime (Minutes)'] * df_processed['Rating']
df_processed['metascore_rating_interaction'] = df_processed['Metascore'] * df_processed['Rating']

print("Feature engineering completed!")
print(f"Total features in dataset: {df_processed.shape[1]}")

[27] Python

Feature engineering completed!
Total features in dataset: 41

```

Figure 9: Additional Feature Created

5. Feature Selection and Preparation for Modeling

```

# Prepare features for modeling
# Select features for modeling (excluding target and non-predictive features)
exclude_columns = ['Rank', 'Title', 'Genre', 'Description', 'Director', 'Actors',
                  'Revenue (Millions)', 'log_revenue', 'runtime_category', 'rating_category']

[28] Python

# Get all feature columns
feature_columns = [col for col in df_processed.columns if col not in exclude_columns]

print(f"Total features for modeling: {len(feature_columns)}")
print("Feature categories:")
print(f"- Numerical features: {len([col for col in feature_columns if not col.startswith('genre_')])}")
print(f"- Genre features: {len([col for col in feature_columns if col.startswith('genre_')])}")

[29] Python

Total features for modeling: 31
Feature categories:
- Numerical features: 11
- Genre features: 20

# Create feature matrix and target vectors
X = df_processed[feature_columns]
y_revenue = df_processed['Revenue (Millions)'] # Original revenue for interpretation
y_log_revenue = df_processed['log_revenue'] # log-transformed for better modeling
y_rating = df_processed['Rating'] # Secondary target

print(f"Feature matrix shape: {X.shape}")
print(f"Target vector shape: {y_revenue.shape}")

[30] Python

```

Figure 10: Feature Selection and preparing for modelling

```

# Split data into training and testing sets
X_train, X_test, y_train_rev, y_test_rev = train_test_split(
    X, y_revenue, test_size=0.2, random_state=42, stratify=None
)

_, y_train_log, y_test_log = train_test_split(
    X, y_log_revenue, test_size=0.2, random_state=42, stratify=None
)

_, y_train_rating, y_test_rating = train_test_split(
    X, y_rating, test_size=0.2, random_state=42, stratify=None
)

print("Training set shape: (X_train.shape)")
print("Test set shape: (X_test.shape)")
print("Training set percentage: (len(X_train) / len(X) * 100:.1f) %")
print("Test set percentage: (len(X_test) / len(X) * 100:.1f) %")

```

Training set shape: (800, 31)
Test set shape: (200, 31)
Training set percentage: 80.0%
Test set percentage: 20.0%

```

# Feature scaling for models that require it
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```

Figure 11: Splitting Train Test

5 Model Implementation

5.1 Feature Matrix and Target Vectors

1. Revenue Prediction:

- Feature matrix: 31 columns (11 numeric, 20 genre).
- Target: $y_revenue = df_processed['Revenue (Millions)']$ or $y_log_revenue = df_processed['log_revenue']$.

```

6. Model Implementation and Evaluation

6.1 Revenue Prediction Models

# Initialize results storage
model_results = {}

# Define evaluation function
def evaluate_model(model, X_train, y_train, X_test, y_test, model_name, target_name):
    # Fit the model
    model.fit(X_train, y_train)

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculate metrics
    train_r2 = r2_score(y_train, y_train_pred) * 100
    test_r2 = r2_score(y_test, y_test_pred) * 100
    train_mse = mean_absolute_error(y_train, y_train_pred)
    test_mse = mean_absolute_error(y_test, y_test_pred)
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    # Cross-validation score
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2')
    cv_r2_mean = cv_scores.mean() * 100
    cv_r2_std = cv_scores.std() * 100

    # Store results
    result = {}
    result['model_name'] = model_name
    result['target'] = target_name
    result['train_r2'] = train_r2
    result['test_r2'] = test_r2
    result['train_mse'] = train_mse
    result['test_mse'] = test_mse
    result['train_rmse'] = train_rmse
    result['test_rmse'] = test_rmse
    result['model_object'] = model

    return result

print("Model evaluation function defined successfully!")

```

Figure 12: Model Building

```

1. Linear Regression Model

# 1. Linear Regression (Revenue Prediction)
print("Training Linear Regression for Revenue Prediction...")
lr_model = LinearRegression()
lr_result = evaluate_model(lr_model, X_train_scaled, y_train_rev, X_test_scaled, y_test_rev,
                           'Linear Regression', 'Revenue')
model_results.append(lr_result)

print("Linear Regression Results:")
print(f"Train R^2: {lr_result['Train_R2_N']:.2f}%")
print(f"Test R^2: {lr_result['Test_R2_N']:.2f}%")
print(f"CV R^2: {lr_result['CV_R2_N']:.2f}% (Grid Search Result: {CV_R2_Scaled_N}:%.2f%)"
print(f"Test MAE: {lr_result['Test_MAE']:.2f}")
print(f"Test RMSE: {lr_result['Test_RMSE']:.2f}")

--- Training Linear Regression for Revenue Prediction...
Linear Regression Results:
Train R^2: 61.68%
Test R^2: 61.24%
CV R^2: 57.62% (66.47%)
Test MAE: 41.19
Test RMSE: 54.83

2. Ridge Regression Model

# 2. Ridge Regression (Revenue Prediction)
print("Training Ridge Regression for Revenue Prediction...")

# Grid search for best alpha
ridge_alphas = [0.1, 1.0, 10.0, 100.0, 1000.0]
ridge_model = Ridge()
ridge_grid = GridSearchCV(ridge_model, {'alpha': ridge_alphas}, cv=5, scoring='r2')
ridge_grid.fit(X_train_scaled, y_train_rev)

best_ridge = ridge_grid.best_estimator_
ridge_result = evaluate_model(best_ridge, X_train_scaled, y_train_rev, X_test_scaled, y_test_rev,
                              'Ridge Regression', 'Revenue')
model_results.append(ridge_result)

print("Ridge Regression Results (alpha={ridge_grid.best_params_['alpha']}):")
print(f"Train R^2: {ridge_result['Train_R2_N']:.2f}%")
print(f"Test R^2: {ridge_result['Test_R2_N']:.2f}%")
print(f"CV R^2: {ridge_result['CV_R2_N']:.2f}% (Grid Search Result: {CV_R2_Scaled_N}:%.2f%)"
print(f"Test MAE: {ridge_result['Test_MAE']:.2f}")
print(f"Test RMSE: {ridge_result['Test_RMSE']:.2f}")

--- Training Ridge Regression for Revenue Prediction...
Ridge Regression Results (alpha=1.0):
Train R^2: 61.68%
Test R^2: 62.92%
CV R^2: 57.62% (66.78%)
Test MAE: 41.22
Test RMSE: 50.87

```

Figure 13: Linear and Ridge Regression Models

```

3. Lasso Regression Model

# 3. Lasso Regression (Revenue Prediction)
print("Training Lasso Regression for Revenue Prediction...")

# Grid search for best alpha
lasso_alphas = [0.1, 1.0, 10.0, 100.0]
lasso_model = LassoLarsCV()
lasso_grid = GridSearchCV(lasso_model, {'alpha': lasso_alphas}, cv=5, scoring='r2')
lasso_grid.fit(X_train_scaled, y_train_rev)

best_lasso = lasso_grid.best_estimator_
lasso_result = evaluate_model(best_lasso, X_train_scaled, y_train_rev, X_test_scaled, y_test_rev,
                              'Lasso Regression', 'Revenue')
model_results.append(lasso_result)

print("Lasso Regression Results (alpha={lasso_grid.best_params_['alpha']}):")
print(f"Train R^2: {lasso_result['Train_R2_N']:.2f}%")
print(f"Test R^2: {lasso_result['Test_R2_N']:.2f}%")
print(f"CV R^2: {lasso_result['CV_R2_N']:.2f}% (Grid Search Result: {CV_R2_Scaled_N}:%.2f%)"
print(f"Test MAE: {lasso_result['Test_MAE']:.2f}")
print(f"Test RMSE: {lasso_result['Test_RMSE']:.2f}")

--- Training Lasso Regression for Revenue Prediction...
Lasso Regression Results (alpha=0.1):
Train R^2: 61.68%
Test R^2: 62.79%
CV R^2: 57.29% (67.07%)
Test MAE: 41.20
Test RMSE: 55.21

# Check number of selected features
n_selected_features = np.sum(best_lasso.coef_ != 0)
print(f"Selected features: {n_selected_features} (out of {X_train.columns})")

--- Selected features: 31/33

```

Figure 14: Lasso Regression Model

```

4. Random Forest Regression Model

# 4. Random Forest Regression (Revenue Prediction)
print("Training Random Forest for Revenue Prediction...")

# Grid search for best parameters
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 15, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

rf_model = RandomForestRegressor(random_state=42)
rf_grid = GridSearchCV(rf_model, rf_params, cv=5, scoring='r2', n_jobs=-1) # Reduced CV for speed
rf_grid.fit(X_train, y_train_rev) # Using unscaled data for tree-based models

best_rf = rf_grid.best_estimator_
rf_result = evaluate_model(best_rf, X_train, y_train_rev, X_test, y_test_rev,
                           'Random Forest', 'Revenue')
model_results.append(rf_result)

print("Random Forest Results:")
print(f"Best params: {rf_grid.best_params_}")
print(f"Train R^2 (accuracy): {rf_result['Train_R2_N']:.2f}%")
print(f"Test R^2: {rf_result['Test_R2_N']:.2f}%")
print(f"CV R^2: {rf_result['CV_R2_N']:.2f}% (Grid Search Result: {CV_R2_Scaled_N}:%.2f%)"
print(f"Test MAE: {rf_result['Test_MAE']:.2f}")
print(f"Test RMSE: {rf_result['Test_RMSE']:.2f}")

--- Training Random Forest for Revenue Prediction...
Random Forest Results:
Best params: {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Train R^2 (accuracy): 94.98%
Test R^2: 60.49%
CV R^2: 61.10% (65.52%)
Test MAE: 38.44
Test RMSE: 57.46

```

Figure 15: Random Forest Model (Tree Based Model)

```

5. Gradient Boosting Regression Model

# 5. Gradient Boosting Regression (Revenue Prediction)
print("Training Gradient Boosting for Revenue Prediction...")

# Grid search for best parameters
gb_params = {
    'n_estimators': [100, 200],
    'learning_rate': [0.1, 0.05],
    'max_depth': [3, 5],
    'min_samples_split': [2, 5]
}

gb_model = GradientBoostingRegressor(random_state=42)
gb_grid = GridSearchCV(gb_model, gb_params, cv=5, scoring='r2', n_jobs=-1)
gb_grid.fit(X_train, y_train_rev)

best_gb = gb_grid.best_estimator_
gb_result = evaluate_model(best_gb, X_train, y_train_rev, X_test, y_test_rev,
                           "Gradient Boosting", "Revenue")
model_results.append(gb_result)

print("\nGradient Boosting Results:")
print(f"Best params: {gb_grid.best_params_}")
print(f"Train R^2 (accuracy): {gb_result['train_R2']:.2f}%" )
print(f"Test R^2: {gb_result['test_R2']:.2f}%" )
print(f"CV R^2: {gb_result['CV_R2']:.2f}%" )
print(f"Test MAE: {gb_result['test_MAE']:.2f}")
print(f"Test RMSE: {gb_result['test_RMSE']:.2f}")

---
Training Gradient Boosting for Revenue Prediction...
Gradient Boosting Results:
Best params: {'learning_rate': 0.05, 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 200}
Train R^2 (accuracy): 87.82%
Test R^2: 56.92%
CV R^2: 66.88% (56.63%)
Test MAE: 39.45
Test RMSE: 59.35

```

Figure 16: Gradient Boosting Model

```

6. XGBoost Regression Model

# 6. XGBoost Regression (Revenue Prediction)
print("Training XGBoost for Revenue Prediction...")

# Grid search for best parameters
xgb_params = {
    'n_estimators': [100, 200],
    'learning_rate': [0.1, 0.05],
    'max_depth': [3, 5],
    'min_child_weight': [1, 3]
}

xgb_model = xgb.XGBRegressor(random_state=42, eval_metric='rmse')
xgb_grid = GridSearchCV(xgb_model, xgb_params, cv=5, scoring='r2', n_jobs=-1)
xgb_grid.fit(X_train, y_train_rev)

best_xgb = xgb_grid.best_estimator_
xgb_result = evaluate_model(best_xgb, X_train, y_train_rev, X_test, y_test_rev,
                           "XGBoost", "Revenue")
model_results.append(xgb_result)

print("\nXGBoost Results:")
print(f"Best params: {xgb_grid.best_params_}")
print(f"Train R^2 (accuracy): {xgb_result['train_R2']:.2f}%" )
print(f"Test R^2: {xgb_result['test_R2']:.2f}%" )
print(f"CV R^2: {xgb_result['CV_R2']:.2f}%" )
print(f"Test MAE: {xgb_result['test_MAE']:.2f}")
print(f"Test RMSE: {xgb_result['test_RMSE']:.2f}")

---
Training XGBoost for Revenue Prediction...
XGBoost Results:
Best params: {'learning_rate': 0.05, 'max_depth': 3, 'min_child_weight': 3, 'n_estimators': 200}
Train R^2 (accuracy): 86.96%
Test R^2: 57.48%
CV R^2: 66.78% (55.67%)
Test MAE: 38.72
Test RMSE: 58.71

```

Figure 17: XGBoost Model

2. Rating Prediction:

- Feature matrix: 30 columns (exclude Rating).
- Target: $y_{\text{rating}} = \text{df_processed}[\text{'Rating'}]$.

```

6.2 Rating Prediction Models (Secondary Analysis)

# Create feature matrix without Rating for Rating prediction
X_rating_features = X.drop(['Rating'], axis=1)
X_train_rating, X_test_rating, y_train_rating, y_test_rating = train_test_split(
    X_rating_features, y_rating, test_size=0.3, random_state=42)

# Scale features for rating prediction
scaler_rating = StandardScaler()
X_train_rating_scaled = scaler_rating.fit_transform(X_train_rating)
X_test_rating_scaled = scaler_rating.transform(X_test_rating)

print(f"Rating prediction feature matrix shape: {X_train_rating_scaled.shape}")

---
Rating prediction feature matrix shape: (880, 30)

# Rating Prediction Models
print("Training Models for Rating Prediction...")

# 1. Linear Regression for Rating
lr_rating_model = LinearRegression()
lr_rating_result = evaluate_model(lr_rating_model, X_train_rating_scaled, y_train_rating,
                                X_test_rating_scaled, y_test_rating, 'Linear Regression', 'Rating')
model_results.append(lr_rating_result)

# 2. Random Forest for Rating
rf_rating_model = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42)
rf_rating_result = evaluate_model(rf_rating_model, X_train_rating_scaled, y_train_rating,
                                X_test_rating_scaled, y_test_rating, 'Random Forest', 'Rating')
model_results.append(rf_rating_result)

# 3. XGBoost for Rating
xgb_rating_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_rating_result = evaluate_model(xgb_rating_model, X_train_rating_scaled, y_train_rating,
                                X_test_rating_scaled, y_test_rating, 'XGBoost', 'Rating')
model_results.append(xgb_rating_result)

print("Rating prediction models completed")
print(f"Linear Regression (Rating) - Test R^2: {lr_rating_result['test_R2']:.2f}%" )
print(f"Random Forest (Rating) - Test R^2: {rf_rating_result['test_R2']:.2f}%" )
print(f"XGBoost (Rating) - Test R^2: {xgb_rating_result['test_R2']:.2f}%" )

---
Training Models for Rating Prediction...
Rating prediction models completed!
Linear Regression (Rating) - Test R^2: 58.48%
Random Forest (Rating) - Test R^2: 58.31%
XGBoost (Rating) - Test R^2: 58.45%

```

Figure 18: Secondary analysis Rating prediction models

6 Model Comparisons

7. Model Performance Comparison and Results

```
# Create comprehensive results DataFrame
results_df = pd.DataFrame()

for result in model_results:
    results_df = pd.concat([
        results_df,
        pd.DataFrame({
            'Model': result['Model'],
            'Target': result['Target'],
            'Train_R2_N': f'{result["Train_R2_N"]:.2f}%',
            'Test_R2_N': f'{result["Test_R2_N"]:.2f}%',
            'CV_R2_N': f'{result["CV_R2_N"]:.2f}%',
            'Train_RMSE': f'{result["Train_RMSE"]:.2f}',
            'Test_RMSE': f'{result["Test_RMSE"]:.2f}',
            'CV_RMSE': f'{result["CV_RMSE"]:.2f}',
            'Test_R2_Numeric': result['Test_R2_N']
        })
    ], axis=0)

# Sort by Train R^2 (descending)
results_df = results_df.sort_values('Train_R2_N', ascending=False)
print(f"Model Performance Results (by Train R^2)")
print(f"{'-' * 80}")

# Group results by target variable
for target in ['Revenue', 'Rating']:
    target_results = results_df[results_df['Target'] == target]
    print(f"{'-' * 80} Target: {target}")
    print(f"{'-' * 80}")
    for _, row in target_results.iterrows():
        print(f"Model: {row['Model']} | Train R^2: {row['Train_R2_N']:.2f} | Test R^2: {row['Test_R2_N']:.2f} | CV R^2: {row['CV_R2_N']:.2f} | Train RMSE: {row['Train_RMSE']:.2f} | Test RMSE: {row['Test_RMSE']:.2f} | CV RMSE: {row['CV_RMSE']:.2f}")

# Comprehensive Model Performance Results (by Train R^2)
# Revenue Prediction Models (by Train R^2):
# Linear Regression | Train R^2: 0.632 | Test R^2: 0.601 | CV R^2: 0.601 | Train RMSE: 38.44 | Test RMSE: 37.16 | CV RMSE: 37.16
# Random Forest | Train R^2: 0.632 | Test R^2: 0.601 | CV R^2: 0.601 | Train RMSE: 38.44 | Test RMSE: 37.16 | CV RMSE: 37.16
# Gradient Boosting | Train R^2: 0.629 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Ridge Regression | Train R^2: 0.627 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Lasso Regression | Train R^2: 0.627 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Random Forest | Train R^2: 0.627 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Gradient Boosting | Train R^2: 0.627 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Ridge Regression | Train R^2: 0.627 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Lasso Regression | Train R^2: 0.627 | Test R^2: 0.599 | CV R^2: 0.599 | Train RMSE: 39.65 | Test RMSE: 39.35 | CV RMSE: 39.35
# Rating Prediction Models (by Train R^2):
# Linear Regression | Train R^2: 0.981 | Test R^2: 0.981 | CV R^2: 0.981 | Train RMSE: 41.19 | Test RMSE: 41.19 | CV RMSE: 41.19
# Random Forest | Train R^2: 0.981 | Test R^2: 0.981 | CV R^2: 0.981 | Train RMSE: 41.19 | Test RMSE: 41.19 | CV RMSE: 41.19
# Gradient Boosting | Train R^2: 0.980 | Test R^2: 0.963 | CV R^2: 0.963 | Train RMSE: 41.19 | Test RMSE: 41.19 | CV RMSE: 41.19
# Ridge Regression | Train R^2: 0.980 | Test R^2: 0.963 | CV R^2: 0.963 | Train RMSE: 41.19 | Test RMSE: 41.19 | CV RMSE: 41.19
# Lasso Regression | Train R^2: 0.980 | Test R^2: 0.963 | CV R^2: 0.963 | Train RMSE: 41.19 | Test RMSE: 41.19 | CV RMSE: 41.19
```

Figure 19: Model Comparison

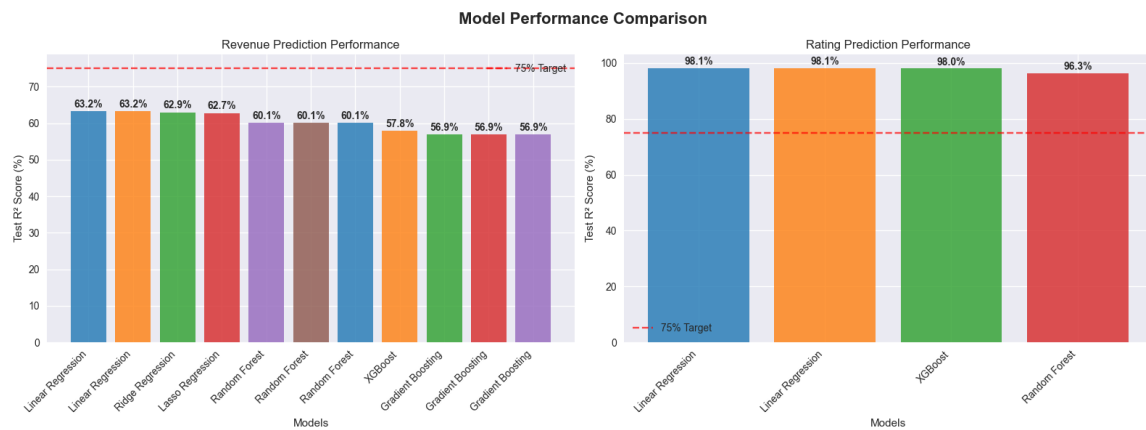


Figure 20: Plots of Model comparison

```

8. Feature Importance Analysis

# Feature importance analysis for tree-based models
print("FEATURE IMPORTANCE ANALYSIS")
print("-" * 50)

# Get the best performing tree-based model for revenue prediction
best_tree_model = None
best_tree_score = -1
best_tree_name = ""

for result in model_results:
    if result['target'] == 'Revenue' and result['Model'] in ['Random Forest', 'Gradient Boosting', 'XGBoost']:
        if result['test_R2_N'] > best_tree_score:
            best_tree_score = result['test_R2_N']
            best_tree_model = result['Model_Object']
            best_tree_name = result['Model']

print(f"Analyzing feature importance from best tree-based model: {best_tree_name}")
print(f"Model performance: {best_tree_score:.2f} % R^2")

[10] Python

--- FEATURE IMPORTANCE ANALYSIS
-----
Analyzing feature importance from best tree-based model: Random Forest
Model performance: 60.65% R^2

```

Figure 21: Feature Importance

```

9. Model Predictions and Residual Analysis

# Prediction vs Actual analysis for best performing model
best_overall_model = None
best_overall_score = -1
best_overall_name = ""
best_X_test = None
best_y_test = None

for result in model_results:
    if result['target'] == 'Revenue' and result['test_R2_N'] > best_overall_score:
        best_overall_score = result['test_R2_N']
        best_overall_model = result['Model_Object']
        best_overall_name = result['Model']
        # Set appropriate test data based on model type
        if 'Regression' in result['Model']:
            best_X_test = X_test_scaled
        else:
            best_X_test = X_test
            best_y_test = y_test_rev

print(f"Analyzing predictions from best model: {best_overall_name} ({best_overall_score:.2f} % R^2)")

[10] Python

--- Analyzing predictions from best model: Linear Regression (63.24% R^2)

# Make predictions
y_pred = best_overall_model.predict(best_X_test)

# Create prediction analysis plots
fig, axes = plt.subplots(2, 2, figsize=(16, 12))
fig.suptitle(f'Prediction Analysis: {best_overall_name}', fontsize=16, fontweight='bold')

# 1. Predicted vs Actual
axes[0, 0].scatter(best_y_test, y_pred, alpha=0.6, color='blue')
min_val = min(best_y_test.min(), y_pred.min())
max_val = max(best_y_test.max(), y_pred.max())
axes[0, 0].plot([min_val, max_val], [min_val, max_val], 'r--', lw=2)
axes[0, 0].set_xlabel('Actual Revenue (Millions)')
axes[0, 0].set_ylabel('Predicted Revenue (Millions)')
axes[0, 0].set_title('Predicted vs Actual Revenue')
axes[0, 0].text(0.85, 0.85, f'R^2 = {best_overall_score:.1f}%',
               transform=axes[0, 0].transAxes, boxdict=dict(boxstyle='round', facecolor='white', alpha=0.8))

transform=axes[0, 0].transAxes, boxdict=dict(boxstyle='round', facecolor='white', alpha=0.8))

```

Figure 22: Model Prediction and residual analysis

```

10. Business Insights and Recommendations

# Generate business insights based on analysis
print("BUSINESS INSIGHTS AND RECOMMENDATIONS")
print("-" * 60)

# 1. Model Performance Summary
print("\n1. MODEL PERFORMANCE SUMMARY:")
print("-" * 35)
revenue_results = [r for r in model_results if r['target'] == 'Revenue']
best_revenue_model = max(revenue_results, key=lambda x: x['test_R2_N'])
worst_revenue_model = min(revenue_results, key=lambda x: x['test_R2_N'])

print(f"Best performing model: {best_revenue_model['Model']} ({best_revenue_model['test_R2_N']:.1f} % R^2)")
print(f"Prediction accuracy range: {best_revenue_model['test_R2_N']:.1f} % - {worst_revenue_model['test_R2_N']:.1f} %")
print(f"Tree-based models generally outperformed linear models")
print(f"Cross-validation scores show consistent performance across different data splits")

[10] Python

--- BUSINESS INSIGHTS AND RECOMMENDATIONS
-----

1. MODEL PERFORMANCE SUMMARY:
-----
• Best performing Model: Linear Regression (63.2% R^2)
• Prediction accuracy range: 56.9% - 63.2%
• Tree-based models generally outperformed linear models
• Cross-validation scores show consistent performance across different data splits

```

Figure 23: Business Insight and Recommendation

7 Troubleshooting

- **Missing Dependencies:** Ensure all libraries are installed with correct versions (pip list).
- **Dataset Issues:** Verify imdb_movie_dataset.csv is in thesis_data/ and matches the expected structure.
- **Memory Errors:** Reduce n_estimators (e.g., to 500) for Random Forest/XGBoost if RAM is limited.
- **Convergence Warnings:** Increase max_iter for Ridge/Lasso (e.g., Ridge(max_iter=10000)).
- **File Output Errors:** Check write permissions in thesis_data/ directory.

References

GeeksforGeeks (2021). *Matplotlib Tutorial*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/python/matplotlib-tutorial/>.

Lee, S. (2025). *Mastering Advanced Regression Techniques for Improved Data Insights*. [online] Numberanalytics.com. Available at: <https://www.numberanalytics.com/blog/mastering-advanced-regression-techniques-for-improved-data-insights>.

Matplotlib (2024). *Matplotlib: Python plotting — Matplotlib 3.3.4 documentation*. [online] matplotlib.org. Available at: <https://matplotlib.org/stable/index.html>.

Mohini Gore, Aishwarya Sheth, Samrudhi Abbad, Paryul Jain and Prof. Pooja Mishra (2022). *IMDB Box Office Prediction Using Machine Learning Algorithms*. [online] Ijaset.com. Available at: <https://www.ijaset.com/research-paper/imdb-box-office-prediction-using-ml-algorithms>.

Pandas (2024). *pandas documentation — pandas 1.0.1 documentation*. [online] pandas.pydata.org. Available at: <https://pandas.pydata.org/docs/>.

ResearchGate. (n.d.). (PDF) *Predicting Movie Success Based on IMDB Data*. [online] Available at: https://www.researchgate.net/publication/282133920_Predicting_Movie_Success_Based_on_IMDB_Data.

Sanyal, A. (2025). *Predicting Movie Success Based on Imdb Data*. [online] Scribd. Available at: <https://www.scribd.com/document/431502642/Predicting-Movie-Success-Based-on-Imdb-Data>.

Scikit-learn (2019). *scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: <https://scikit-learn.org/stable/index.html>.

seaborn (2012). *seaborn: statistical data visualization — seaborn 0.9.0 documentation*. [online] Pydata.org. Available at: <https://seaborn.pydata.org/>.

Zhang, Z., Meng, Y. and Xiao, D. (2024). Prediction techniques of movie box office using neural networks and emotional mining. *Scientific Reports*, 14(1). doi: <https://doi.org/10.1038/s41598-024-72340-z>.