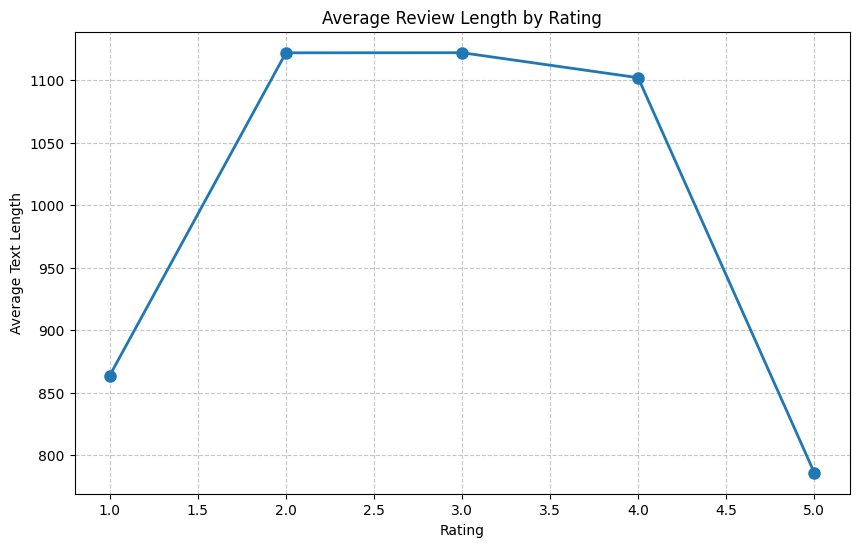
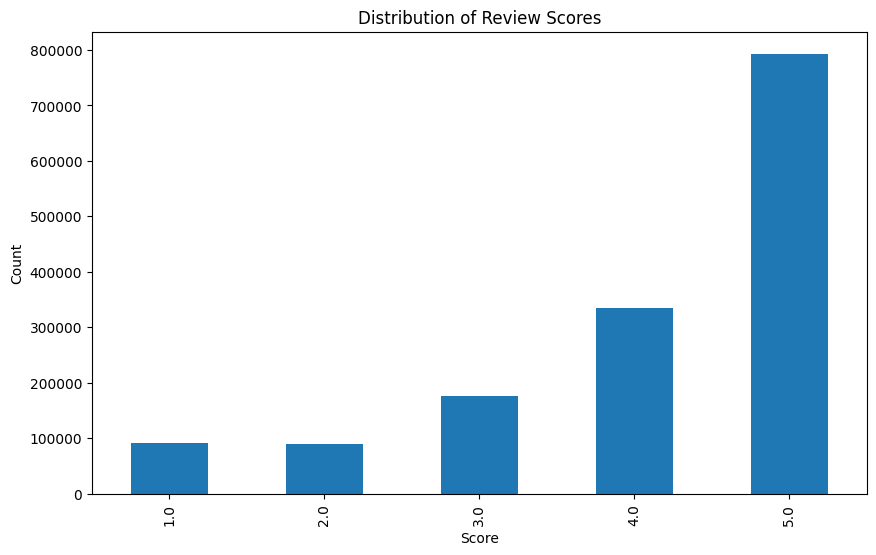
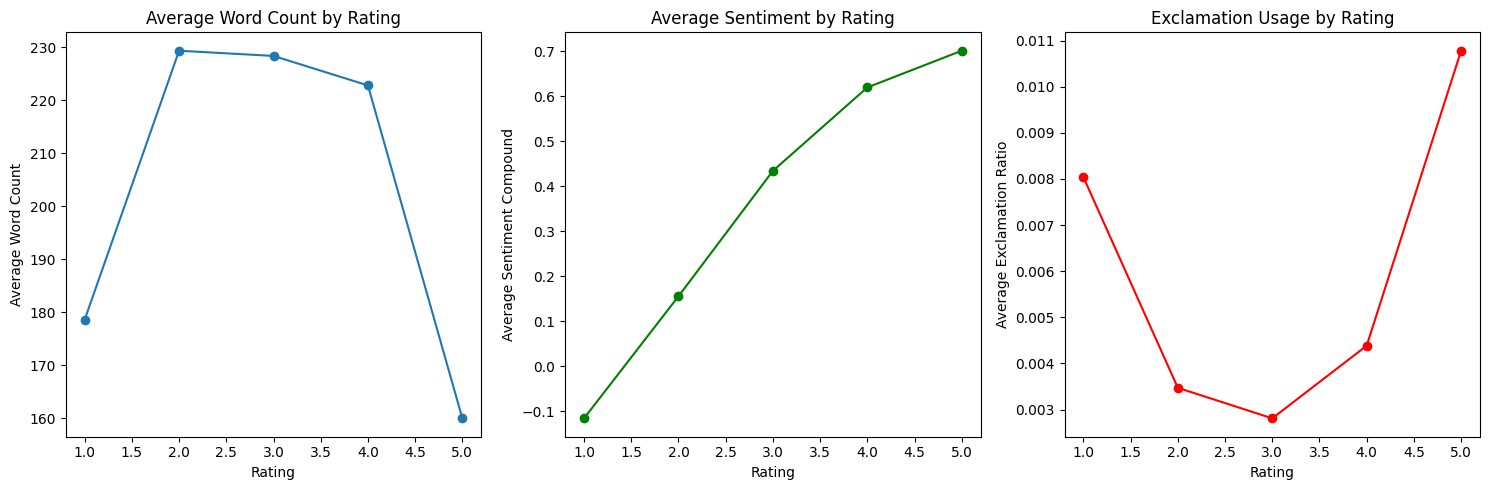
# **Midterm: Movie Review Classification Report**

## **1. Data Observation**

Looking at the dataset visualization, several key patterns emerge. The distribution of review scores shows a clear bias towards 5-star ratings, with approximately 780,000 5-star reviews, followed by about 330,000 4-star reviews. Lower ratings (1-3 stars) appear less frequently, with each having less than 200,000 reviews, indicating a significant class imbalance in our dataset.

The relationship between ratings and text characteristics reveals interesting patterns:

1. Word Count: Reviews with 2-star ratings tend to be the longest (around 230 words), while 5-star reviews are notably shorter (about 160 words), suggesting that dissatisfied customers write more detailed reviews.
2. Sentiment: There's a strong positive correlation between rating and sentiment scores, rising steadily from -0.1 for 1-star reviews to 0.7 for 5-star reviews.
3. Exclamation Usage: Exclamation marks appear most frequently in 5-star reviews (0.011 ratio), with a second peak in 1-star reviews (0.008 ratio), indicating stronger emotional expression in extreme ratings.

This imbalanced distribution and clear patterns in text characteristics influenced our decision to use SMOTE and undersampling techniques in our model development.

## 2. Libraries Used In This Notebook

In this notebook, I relied on several key Python libraries to handle different aspects of the data processing and analysis pipeline. For my basic data processing needs, I employed NumPy and Pandas. To visualize the data, I integrated Matplotlib and Seaborn into my workflow. In order to perform text processing and Natural Language Processing (NLP), I implemented NLTK (Natural Language Toolkit) as my primary tool. Within NLTK, I specifically utilized its stopwords corpus for removing common words, word\_tokenize and sent\_tokenize for breaking down text into words and sentences, and the SentimentIntensityAnalyzer for extracting sentiment scores from reviews. I also incorporated the TfidfVectorizer from scikit-learn's feature extraction module to convert text data into numerical features.

For my machine learning implementation, I heavily relied on scikit-learn's comprehensive suite of tools. I used train\_test\_split, cross\_val\_score, KFold, and StratifiedKFold for model selection and evaluation. For assessing model performance, I implemented accuracy\_score, confusion\_matrix, and classification\_report. I also utilized StandardScaler for feature normalization. In terms of models, I experimented with RandomForestClassifier, GradientBoostingClassifier, LogisticRegression, and LinearSVC to find the best performing algorithm for my prediction task.

To handle the challenge of imbalanced data in my dataset, I implemented specialized tools from the imbalanced-learn library, including SMOTE for oversampling and RandomUnderSampler for undersampling, which I combined using the Pipeline class to create a balanced training approach.

Finally, to optimize the performance of my text processing operations, I incorporated the pandarallel library, which allowed me to parallelize my Pandas operations and significantly reduce processing time when handling large volumes of text data. I made sure to set random seeds across all relevant libraries to ensure reproducibility of my results.

## 3. Features of The Dataset

In the feature engineering process, I implemented several additional features to enhance the predictive power of my model. To start, the dataset has many duplicate user Id and product Id so I created 'user\_avg\_score' and 'user\_review\_count' to capture individual user rating patterns and experience levels. Similarly, for product analysis, I added 'product\_avg\_score' and 'product\_review\_count' to understand product performance history.

In terms of text analysis, I added basic metrics like 'text\_word\_count', 'text\_avg\_word\_length', 'text\_sentence\_count', and 'text\_avg\_sentence\_length'. For deeper text insights, I calculated 'text\_long\_words\_ratio' to measure vocabulary complexity. The sentiment analysis features ('text\_sentiment\_neg', 'text\_sentiment\_pos', 'text\_sentiment\_neu', 'text\_sentiment\_compound') were extracted using VADER to capture emotional content.

I kept 'helpfulness\_ratio' (HelpfulnessNumerator/HelpfulnessDenominator) from the starter code. I also wanted to measure writing style markers such as 'text\_exclamation\_ratio', 'text\_question\_ratio', and 'text\_caps\_ratio'. Summary features ('summary\_word\_count', 'summary\_avg\_word\_length', 'summary\_sentiment\_compound') were added to capture the concise version of user opinions.

## 4. The Model

In my model selection process, I focused on two primary algorithms: Random Forest Classifier and Logistic Regression, specifically chosen for their efficiency and performance characteristics with our large, text-heavy dataset. While LinearSVC and GradientBoosting were initially considered, they were ultimately excluded due to their inability to support parallel processing (n\_jobs=-1), which would have significantly increased computation time given our dataset size.

Random Forest emerged as the best performer, which makes sense given our feature set. Its ability to handle non-linear relationships, manage high-dimensional data effectively, and provide feature importance rankings made it particularly suitable for our mix of text-based and numerical features. I configured it with 200 trees, a max depth of 10, and balanced class weights to handle our imbalanced rating distribution.

Logistic Regression served as our baseline model, chosen for its interpretability and computational efficiency. Despite being a simpler model, it handled the multi-class nature of our rating prediction task well. I set it with a max iteration of 1000 and balanced class weights to ensure convergence and fair prediction across all rating classes.

Both models were implemented within a pipeline that included SMOTE for oversampling and RandomUnderSampler for undersampling to address the significant class imbalance we observed in our rating distribution. The parallel processing capability (n\_jobs=-1) of both models was crucial for efficient training on our large dataset.

## 5. Conclusion and Future Improvements

Through this project, I learned several key insights about movie review classification. The strong correlation between sentiment scores and ratings validates the effectiveness of sentiment analysis in review classification. I also learn to better understand and pre-process the dataset.

The model's performance could be improved through several enhancements in feature engineering. I wanted to implement more advanced text features like n-grams or word embeddings, add time-based features to capture temporal patterns in ratings, and create interaction features between user and product statistics. But ultimately I didn't, since I wasn't familiar with those processes.

From a modeling perspective, I could achieve better performance by fine-tuning Random Forest hyperparameters through more extensive GridSearchCV, experimenting with ensemble methods combining Random Forest and Logistic Regression, and trying other tree-based algorithms like XGBoost with CPU-only configuration.

The data processing pipeline could also be refined by exploring different ratios for SMOTE and undersampling, implementing more sophisticated text preprocessing techniques, and considering cross-validation for feature selection. A more balanced validation strategy could also be developed to better handle the significant class imbalance in our dataset.