Automated Review Rating System

1. Project Overview

The project **Automated Review Rating System** aims to develop a smart framework capable of predicting star ratings from customer reviews. Using natural language processing techniques, the system interprets textual feedback to identify sentiment and linguistic patterns. Machine learning models are trained on labeled review data to establish relationships between customer opinions and corresponding rating levels. By evaluating performance across different dataset distributions, the system demonstrates the effectiveness of AI-driven solutions in providing consistent, scalable, and efficient review analysis.

2. Environment Setup

The project was developed and executed in **Jupyter Notebook**, providing an interactive environment for code execution and analysis. The implementation was carried out using **Python 3.10**, along with several essential libraries for data preprocessing, visualization, and machine learning.

The following libraries were installed and utilized:

- **pandas** – for dataset loading, manipulation, and analysis.
- **numpy** – for numerical computations and efficient array handling.
- **matplotlib** & **seaborn** – for data visualization and graphical representation of trends.
- **scikit-learn** – for feature extraction, model training, evaluation, and performance metrics.
- **spaCy** – for natural language processing and lemmatization.
- **re** – for text pattern recognition and cleaning using regular expressions.

3. GitHub Project Setup

A GitHub repository named Automated-Review-Rating-System was created to manage the project's code, notebooks, and resources. The repository provides version control, collaboration capabilities, and a centralized location for all project files.The GitHub Repository Setup is illustrated in Figure 1.

3.1 Directory Structure

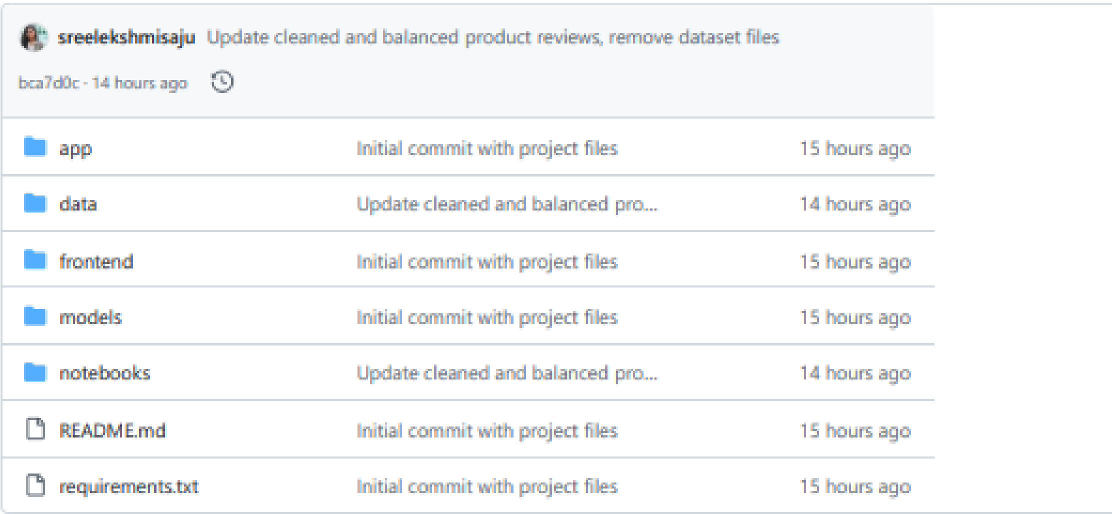


Figure_1: GitHub Repository Setup

4. Data Collection

The project utilizes publicly available datasets from Kaggle to train and evaluate the Automated Review Rating System. The datasets collected include:

- **Amazon kindle_reviews.csv** – contains 71221 reviews across 10 columns. Dataset Link

After preprocessing, the final cleaned dataset contains 71,217 reviews. The distribution of ratings in the merged_reviews.csv dataset is as follows:

Rating	Count
1	3786
2	5435
3	6000
4	19999
5	35997

Table 1: Distribution of ratings in the merged dataset

The cleaned and the new dataset is available at the following link: New Dataset Link

This new dataset provides a robust foundation for training machine learning models to predict review ratings.

4.1 Imbalanced Dataset

To simulate real-world scenarios where some review ratings are more frequent than others, an imbalanced dataset was created from the remaining reviews after removing those included in the balanced dataset. This approach ensures that evaluation reflects practical challenges where certain classes dominate the data.

First, all rows used in the balanced dataset were excluded from the original cleaned dataset to avoid duplication. The remaining dataset formed the pool from which the imbalanced dataset was sampled.

Next, target proportions for each rating class were defined to reflect typical distributions observed in real review datasets. In this study, the proportions were set as 10% for rating 1, 15% for rating 2, 25% for rating 3, 30% for rating 4, and 20% for rating 5.

Sampling was performed without replacement, ensuring that no duplicate reviews were included. For each rating category, rows were selected according to the defined ratio, while respecting the availability of data in each class. Classes with fewer remaining reviews than required were sampled up to the available number of rows.

Finally, all sampled rows from each rating class were combined and shuffled to form the final imbalanced dataset. The distribution of reviews across ratings in the imbalanced dataset is as follows:

- Rating 4: 17,499
- Rating 5: 11,743
- Rating 3: 3,500
- Rating 2: 2,935
- Rating 1: 1,286

The final imbalanced dataset contains 36,963 reviews across these five rating categories. This dataset allows for testing model performance under realistic conditions, where some rating categories are underrepresented, thereby providing insights into model robustness and sensitivity to class imbalance.

The dataset is available at the following link: Imbalanced Dataset Link

Imbalanced Dataset Details:

```
Imbalanced Dataset Distribution (no duplicates):
 Rating
4    17499
5    11743
3     3500
2     2935
1     1286
Name: count, dtype: int64
Imbalanced dataset shape: (36963, 3)
```

5. Data Preprocessing

Data preprocessing is a critical step in preparing raw text data for machine learning. It involves cleaning, standardizing, and transforming the data to ensure it is consistent, meaningful, and suitable for analysis. These steps help improve model accuracy and reliability by removing noise, normalizing text, and structuring the reviews for effective feature extraction.

5.1 Column Renaming

Column names are standardized to improve clarity and make the dataset easier to work with. For example, columns containing review text and ratings are renamed to `Review` and `Rating`, respectively. Standardized column names help avoid confusion during analysis and make the code and documentation more readable and consistent. This step ensures uniformity across different datasets, facilitating seamless merging and preprocessing operations.

The following Python code was used to rename dataset columns for consistency:

Python Code

```
# Rename columns for better readability and consistency
df.rename(columns={'reviewText': 'Review', 'overall': 'Rating'},
   inplace=True)
print("Columns after renaming:\n", df.columns.tolist())
```

Output:

```
Columns after renaming:
 ['Unnamed: 0', 'asin', 'helpful', 'Rating', 'Review', 'reviewTime', 'reviewerID', 'reviewerName', 'summary', 'unixReviewTime']
```

5.2 Handling Missing Values

Identifying and handling missing data is a crucial step in preparing a clean dataset for machine learning. Missing values can introduce bias, reduce model performance, and cause errors during analysis. In this project, the number of missing values in each column was calculated to assess data completeness and determine the necessary preprocessing actions.

The following Python code was used to quantify missing data in the dataset:

Python Code

```
# Count missing values in each column
missing_data = df.isnull().sum()
print("\nMissing Data Count per Column:")
print(missing_data)
# Total missing values in the dataset
total_missing = missing_data.sum()
print("\nTotal Missing Values in Dataset:", total_missing)
```

This step provides an overview of data quality by reporting both per-column missing values and the total missing entries in the dataset. Understanding the extent of missing data is essential for deciding whether to remove, impute, or otherwise handle incomplete entries before model training.

Output:

```
Missing Data Count per Column:
Unnamed: 0          0
asin                0
helpful             0
Rating              0
Review              2
reviewTime          0
reviewerID          0
reviewerName      577
summary            15
unixReviewTime      0
dtype: int64

Total Missing Values in Dataset: 594
```

5.3 Handling Missing Data

To ensure the integrity of the dataset and avoid errors during model training, rows containing missing values in critical columns were removed. Specifically, entries with missing values in the `Review` or `Rating` columns were identified and dropped. After removal, the dataset index was reset to maintain sequential ordering and prevent indexing issues during subsequent processing steps.

The following Python code was used to clean and verify the dataset:

Python Code

```python
# Count missing values in 'Review' and 'Rating'
print("Missing values in Review and Rating before cleaning:")
print(df[['Review', 'Rating']].isnull().sum())
# Drop rows with missing 'Review' or 'Rating'
df.dropna(subset=['Review', 'Rating'], inplace=True)
df.reset_index(drop=True, inplace=True)
print("\nAfter handling missing data:")
print(df[['Review', 'Rating']].isnull().sum())
print("Dataset shape:", df.shape)
print("\nDataset Information:")
df.info()
df.head()
```

This step ensures that the dataset contains only complete records, thereby improving the reliability of feature extraction and model training. It also provides a clear understanding of the dataset's structure and size after cleaning, which is essential for accurate performance evaluation.

Output:

```
Missing values in Review and Rating before cleaning:
Review     2
Rating     0
dtype: int64

After handling missing data:
Review     0
Rating     0
dtype: int64
Dataset shape: (71219, 10)
```

5.4 Count Number of duplicates

Duplicate entries in a dataset can introduce bias and inflate the importance of certain observations, potentially affecting the performance of machine learning models. In the Automated Review Rating System, duplicate reviews with identical ratings were identified to ensure data integrity and avoid redundancy in training data.

The following Python code was used to detect duplicate records based on the `Review` and `Rating` columns:

Python Code

```
# Count duplicates based on 'Review' and 'Rating'
duplicate_count = df.duplicated(subset=['Review', 'Rating']).sum()
print(f"Number of duplicate rows in dataset: {duplicate_count}")
duplicates = df[df.duplicated(subset=['Review', 'Rating'], keep=False)]
print("\nSample duplicate rows:")
duplicates.head()
```

This process provides an overview of redundancy within the dataset, allowing for informed decisions on whether to remove duplicates. Handling duplicates is critical for training a robust model that generalizes well across unique customer reviews.

Output:

```
Number of duplicate rows in dataset: 2
```

5.5 Removing Duplicate and Conflicting Records

To maintain the quality and reliability of the dataset, exact duplicates and conflicting entries were removed. Exact duplicates, where both the `Review` text and `Rating` were identical, were dropped while keeping the first occurrence. Additionally, conflicting reviews, where the same review text had different ratings, were identified and removed to prevent contradictory information from affecting model training. After these removals, the dataset index was reset to ensure sequential ordering for smooth processing.

The following Python code was used to perform these operations:

Python Code

```
# Remove exact duplicates (same Review + Rating)
df = df.drop_duplicates(subset=['Review', 'Rating'], keep='first')
# Remove conflicting reviews (same Review text, different Ratings)
conflict_idx = df[df.duplicated(subset=['Review'], keep=False) &
                  ~df.duplicated(subset=['Review', 'Rating'], keep=False)
                  ].index
df.drop(index=conflict_idx, inplace=True)
# Reset index after removals
df.reset_index(drop=True, inplace=True)
print("After removing duplicates and conflicts, dataset shape:", df.
    shape)
df.head()
```

By removing duplicate and conflicting records, the dataset is refined to contain only unique and consistent entries. This step ensures that the machine learning models learn meaningful patterns from authentic and reliable data, thereby improving prediction accuracy.

Output:

```
After removing duplicates and conflicts, dataset shape: (71217, 10)
```

5.6 Selecting Relevant Columns

To streamline the dataset for model training, only the essential columns, `Rating` and `Review`, were retained. This step eliminates unnecessary information that does not contribute to the predictive task, thereby simplifying the dataset and reducing computational overhead during feature extraction and model training.

The following Python code was used to select the relevant columns:

Python Code

```
# Keep only 'Rating' and 'Review' columns for modeling
df = df[['Rating', 'Review']]
print("Final dataset columns:", df.columns.tolist())
print("Final dataset shape:", df.shape)
df.head()
```

By focusing on only the necessary features, the preprocessing pipeline ensures that the machine learning models operate efficiently and learn patterns directly related to predicting customer review ratings. This step also lays the foundation for consistent and structured input to subsequent text processing and modeling stages.

Output:

```
Final dataset columns: ['Rating', 'Review']
Final dataset shape: (71217, 2)
```

5.7 Grouping and Aggregation

Understanding the distribution of ratings within the dataset is a crucial step in exploratory data analysis. It provides insights into class balance, potential biases, and the overall sentiment trends expressed by customers. In this project, the count of reviews per rating, the average rating, and the minimum and maximum ratings were calculated to summarize the dataset's characteristics.

The following Python code was used to analyze the rating distribution:

Python Code

```python
# Count of reviews per rating
review_count_per_rating = df.groupby('Rating')['Review'].count()
print("Number of Reviews per Rating:\n", review_count_per_rating)
# Average rating
average_rating = df['Rating'].mean()
print("\nAverage Rating of Dataset:", round(average_rating, 2))
# Minimum and Maximum rating
min_rating = df['Rating'].min()
max_rating = df['Rating'].max()
print("Rating Range: {} to {}".format(min_rating, max_rating))
```

This analysis revealed the number of reviews corresponding to each rating level, the overall average rating, and the rating range. Such insights are important for identifying imbalances in the dataset, which can inform subsequent steps such as creating balanced datasets and selecting appropriate machine learning strategies.

Output:

```
Number of Reviews per Rating:
 Rating
1      3786
2      5435
3      6000
4     19999
5     35997
Name: Review, dtype: int64

Average Rating of Dataset: 4.11
Rating Range: 1 to 5
```

5.8 Cleaning text - Remove emojis, Special characters and Symbols

Text data often contains noise such as punctuation, special characters, emojis, and inconsistent capitalization, which can negatively impact machine learning models. To address this, a text cleaning function was implemented to standardize the review content. The cleaning process includes converting all text to lowercase, removing punctuation and symbols, eliminating emojis and non-ASCII characters, and reducing extra whitespace. These steps help in normalizing the textual data, making it suitable for feature extraction and model training.

The following Python code was used to clean and normalize the review text:

Python Code

```
import string
import re
# Function to clean review text
def clean_text(text):
    # Convert to string in case of missing or non-string entries
    text = str(text)
    # Remove emojis and special unicode characters
    text = text.encode('ascii', 'ignore').decode('ascii')
    # Remove punctuation and symbols
    text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)
    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Convert to lowercase
    text = text.lower()
    return text
# Apply cleaning function to 'Review' column
df['Clean_Review'] = df['Review'].apply(clean_text)
df[['Review', 'Clean_Review']].head()
```

By performing text cleaning, the reviews are transformed into a standardized and simplified format. This improves the quality of input for natural language processing techniques and ensures that the machine learning models can focus on meaningful linguistic patterns rather than noise.

Output:

```
Sample cleaned reviews:
```

	Review	Clean_Review
0	This was my first IR book I've read and it jus...	this was my first ir book ive read and it just...
1	In this book Holly is looking for her mother w...	in this book holly is looking for her mother w...
2	Here we have a story about Keli and Phillip Go...	here we have a story about keli and phillip go...
3	I very much enjoyed reading about the Italian ...	i very much enjoyed reading about the italian ...
4	Readers may be attracted by the curious title ...	readers may be attracted by the curious title ...

5.9 Detection of Consecutive Repeated Words

In textual data, consecutive repeated words can occur due to typing errors or emphasis, which may introduce noise and affect model performance. Identifying these repeated words is an important preprocessing step to clean the text and improve feature quality. In this project, a function was implemented to detect consecutive duplicate words using regular expressions. The function searches for any word that appears twice in succession, ignoring case, and returns all occurrences for further inspection.

The following Python code was used to identify repeated words in the dataset:

Python Code

```
import re
# Function to find consecutive duplicate words
def find_repeated_words(text):
    # Regex pattern: captures consecutive duplicate words, ignoring case
    pattern = r'\b(\w+)\s+\1\b'
    # Find all matches, preserve original casing
    matches = re.findall(pattern, text, flags=re.IGNORECASE)
    # Return list of tuples like [(word, word)]
    return [(match, match) for match in matches]
# Apply function to dataset
df['Repeated_Words'] = df['Review'].apply(find_repeated_words)
print("Sample reviews with consecutive repeated words:")
df[df['Repeated_Words'].str.len() > 0][['Review', 'Repeated_Words']].
    head()
```

By detecting consecutive repeated words, the preprocessing pipeline can address potential noise in the dataset. This step ensures that the textual data fed to machine learning models is more consistent and representative of actual customer feedback, enhancing the reliability of sentiment and rating predictions.

Output:

```
Sample reviews with consecutive repeated words:
```

	Review	Repeated_Words
19	I thought I was going to get more of a story t...	[(the, the)]
78	I picked this short story up because it was fr...	[(FUN, FUN)]
82	It takes a great amount of time to read becaus...	[(need, need)]
135	This book has everything you need. Well writt...	[(much, much)]
138	I think all your book are great Jen. This one ...	[(about, about)]

5.10 Removing Consecutive Repeated Words

Consecutive repeated words can introduce redundancy and noise into textual data, potentially affecting the learning process of machine learning models. To enhance text quality, repeated words were removed by replacing consecutive duplicates with a single occurrence. This normalization step ensures that the review content is more concise and accurately represents the customer's sentiment.

The following Python code was used to remove consecutive repeated words:

Python Code

```
# Function to remove consecutive repeated words
def remove_repeated_words(text):
    # Regex: replace consecutive duplicate words with a single
       occurrence
    pattern = r'\b(\w+)(\s+\1\b)+'
    return re.sub(pattern, r'\1', text, flags=re.IGNORECASE)
df['Clean_Review'] = df['Clean_Review'].apply(remove_repeated_words)
print("Sample reviews after removing consecutive repeated words:")
df[['Review', 'Clean_Review']].head()
```

By removing consecutive repeated words, the dataset becomes cleaner and more consistent. This preprocessing step improves the quality of input for feature extraction, helping the machine learning models focus on meaningful linguistic patterns rather than redundant information.

Output:

Sample reviews after removing consecutive repeated words:

	Review	Clean_Review
0	This was my first IR book I've read and it jus...	this was my first ir book ive read and it just...
1	In this book Holly is looking for her mother w...	in this book holly is looking for her mother w...
2	Here we have a story about Keli and Phillip Go...	here we have a story about keli and phillip go...
3	I very much enjoyed reading about the Italian ...	i very much enjoyed reading about the italian ...
4	Readers may be attracted by the curious title ...	readers may be attracted by the curious title ...

5.11 Sorting Dataset by Rating

Sorting the dataset based on rating values provides a structured view of reviews from the lowest to highest ratings. This organization facilitates better understanding of rating distribution, enables easier inspection of extreme cases, and can assist in subsequent analysis or sampling strategies for Imbalanced datasets.

The following Python code was used to sort the dataset by the `Rating` column:

Python Code

```
# Sort dataset by Rating in ascending order
df_sorted = df.sort_values(by='Rating', ascending=True).reset_index(
    drop=True)
print("Dataset sorted by Rating:")
df_sorted.head()
```

By arranging reviews in ascending order of ratings, the dataset becomes more interpretable and allows for systematic examination of patterns across different rating levels. This step also aids in the preparation of Imbalanced datasets for model training.

Output:

Dataset sorted by Rating:

	Rating	Review	Clean_Review
0	1	If you're looking for a clean book this is NOT...	if youre looking for a clean book this is not ...
1	1	This could have perhaps been a good novel ha i...	this could have perhaps been a good novel ha i...
2	1	Even if you get it free and put it on your Kin...	even if you get it free and put it on your kin...
3	1	Not sure what book the others read but I didn'...	not sure what book the others read but i didnt...
4	1	This did nothing for me. Sorry, but if I had t...	this did nothing for me sorry but if i had to ...

5.12 Text Normalization: Lowercasing

To ensure uniformity in textual data, all review text was converted to lowercase. This normalization step helps in reducing redundancy caused by variations in letter casing, allowing the model to treat words like "Good" and "good" as the same token. Consistent text representation is essential for accurate feature extraction and improves the performance of machine learning models on natural language data.

The following Python code was used to apply lowercasing to all reviews:

Python Code

```
# Convert all review text to lowercase
df['Review'] = df['Review'].str.lower()
df.head()
```

Output:

	Rating	Review
0	1	this book was so short it wouldn't even qualif...
1	1	i can't believe all the good reviews for this ...
2	1	loved the book. great mystery and love story. ...
3	1	like another reviewer i found this book creepy...
4	1	this book was a real let down. gets into peopl...

5.13 Removing URLs from Reviews

Customer reviews may contain URLs linking to external content, which are generally irrelevant for sentiment or rating prediction. To prevent noise and improve model performance, all URLs were removed from the review text using regular expressions. This step ensures that only meaningful textual content is retained for feature extraction and modeling.

The following Python code was used to remove URLs from the reviews:

Python Code

```
import re
# Remove URLs from the review text
df['Review'] = df['Review'].apply(lambda x: re.sub(r'http\S+|www\S+|
    https\S+', '', x, flags=re.IGNORECASE))
df.head()
```

By eliminating URLs, the dataset becomes cleaner and more focused on the actual customer opinions, allowing machine learning models to learn patterns from relevant textual content without distraction from extraneous links.

Output:

	Rating	Review
0	1	this book was so short it wouldn't even qualif...
1	1	i can't believe all the good reviews for this ...
2	1	loved the book. great mystery and love story. ...
3	1	like another reviewer i found this book creepy...
4	1	this book was a real let down. gets into peopl...

5.14 Removing HTML Tags

Customer reviews may occasionally contain HTML tags, which do not contribute to the semantic meaning of the text and can introduce noise in the dataset. To ensure clean and meaningful textual data, all HTML tags were removed from the reviews using regular expressions. This step improves the quality of input for natural language processing and feature extraction.

The following Python code was used to remove HTML tags from the reviews:

Python Code

```
# Remove HTML tags from review text
df['Review'] = df['Review'].apply(lambda x: re.sub(r'<.*?>', '', x))
df.head()
```

By stripping HTML tags, the dataset becomes more consistent and focused solely on the textual content, which enhances the effectiveness of machine learning models in predicting review ratings accurately.

Output:

	Rating	Review
0	1	this book was so short it wouldn't even qualif...
1	1	i can't believe all the good reviews for this ...
2	1	loved the book. great mystery and love story. ...
3	1	like another reviewer i found this book creepy...
4	1	this book was a real let down. gets into peopl...

5.15 Removing Emojis, Punctuation, and Special Characters

Text data often contains emojis, punctuation marks, and other special characters that do not contribute meaningfully to sentiment or rating prediction. To enhance the quality of the dataset, all non-alphanumeric characters were removed from the reviews. Additionally, extra whitespaces were eliminated to ensure uniform spacing between words. These preprocessing steps help create clean and consistent textual data for feature extraction and machine learning.

The following Python code was used to remove unwanted characters and normalize spacing:

Python Code

```
import re
# Remove emojis, punctuation, and special characters
df['Review'] = df['Review'].apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '
    ', x))
# Remove extra whitespaces
df['Review'] = df['Review'].apply(lambda x: re.sub(r'\s+', ' ', x).
    strip())
df.head()
```

By cleaning the text of unnecessary symbols and normalizing whitespace, the dataset is better prepared for downstream natural language processing tasks, improving the reliability and performance of machine learning models.

Output:

	Rating	Review
0	1	this book was so short it wouldnt even qualify...
1	1	i cant believe all the good reviews for this b...
2	1	loved the book great mystery and love story wo...
3	1	like another reviewer i found this book creepy...
4	1	this book was a real let down gets into people...

5.16 Stopword Identification and Analysis

Stopwords are common words such as “the,” “is,” and “and” that carry minimal semantic meaning and can often be removed to reduce noise in text data. In this project, the English stopwords list provided by SpaCy was used to identify and analyze the presence of stopwords in customer reviews. Each review was evaluated to count the number of stopwords it contains, providing insight into the textual composition and helping guide further preprocessing steps such as stopword removal.

The following Python code was used to identify and count stopwords in the dataset:

Python Code

```python
import spacy
# Load SpaCy English model
nlp = spacy.load('en_core_web_sm')
# Get stopwords list
stopwords = nlp.Defaults.stop_words
print(f"Number of stopwords: {len(stopwords)}")
print("Sample stopwords:", list(stopwords)[:20])
# Count stopwords in each review
df['Stopword_Count'] = df['Review'].apply(lambda x: sum(1 for token in
    x.split() if token in stopwords))
df[['Review', 'Stopword_Count']].head()
```

Analyzing stopwords helps in understanding the structure of the reviews and informs decisions on removing non-informative words to improve the quality of features for machine learning models. This step contributes to a more focused and semantically meaningful representation of the review text.

Output:

```
Number of stopwords: 326
Sample stopwords: ['in', 'mine', 'whom', 'themselves', 'on', 'well', 'either', 'through', 'though', 'someone', 'ca', 'everyone', 'about', 'ou
t', 'become', 'all', 'everywhere', 'should', 'unless', 'where']
```

	Review	Stopword_Count
0	this book was so short it wouldnt even qualify...	15
1	i cant believe all the good reviews for this b...	25
2	loved the book great mystery and love story wo...	12
3	like another reviewer i found this book creepy...	18
4	this book was a real let down gets into people...	13

5.17 Total Stopwords Analysis

Quantifying the total number of stopwords across the dataset provides a broader understanding of how prevalent non-informative words are in the reviews. This analysis helps evaluate the potential impact of stopwords on feature extraction and model performance, and informs whether additional preprocessing, such as stopword removal, is necessary.

The following Python code was used to calculate the total number of stopwords in the dataset:

Python Code

```
# Total stopwords in the entire dataset
total_stopwords = df['Stopword_Count'].sum()
print("Total stopwords in all reviews:", total_stopwords)
```

By analyzing the total stopwords, the preprocessing pipeline gains insight into the textual composition of the reviews. Removing or handling stopwords appropriately can enhance the quality of features and improve the accuracy and efficiency of machine learning models for review rating prediction.

Output:

```
Total stopwords in all reviews: 2280723
```

5.18 Stopword Removal

To reduce noise and focus on the meaningful content of reviews, stopwords were removed from the text. Stopwords, such as "the," "is," and "and," generally do not contribute to the sentiment or rating prediction and can dilute feature significance. By eliminating these non-informative words, the dataset is transformed into a more concise and semantically rich representation suitable for machine learning.

The following Python code was used to remove stopwords from the reviews:

Python Code

```
# Function to remove stopwords
def remove_stopwords(text):
    doc = nlp(text)
    return ' '.join([token.text for token in doc if token.text not in
        nlp.Defaults.stop_words])
# Apply to the Review column
df['Review'] = df['Review'].apply(remove_stopwords)
df.head()
```

Removing stopwords improves the quality of input for feature extraction techniques, such as TF-IDF or word embeddings, ensuring that the models learn from words that carry meaningful information relevant to predicting customer ratings.

Output:

	Rating	Review	Stopword_Count
0	1	book short nt qualify novella guess pay free	15
1	1	nt believe good reviews book like taking lesso...	25
2	1	loved book great mystery love story read books...	12
3	1	like reviewer found book creepy repulsed bacch...	18
4	1	book real let gets people turning animals biza...	13

5.19 Choice of spaCy for NLP Tasks

In this project, **spaCy** was chosen over **NLTK** for natural language processing tasks such as tokenization, stopword handling, and lemmatization. While NLTK provides a broad range of NLP tools, spaCy is optimized for industrial-scale applications, offering faster processing speeds and more efficient memory usage.

SpaCy also provides pretrained models that support part-of-speech tagging, dependency parsing, and named entity recognition with high accuracy. Additionally, spaCy's pipeline design allows seamless integration of multiple preprocessing steps, making it more suitable for handling large datasets of customer reviews.

The choice of spaCy ensures that the text preprocessing pipeline is both efficient and scalable, improving the overall performance of machine learning models.

5.20 Lemmatization

Lemmatization is the process of reducing words to their base or root form, ensuring that different inflections of the same word are treated uniformly. For example, "running" and "ran" are both reduced to "run". This normalization step reduces feature dimensionality and improves the ability of machine learning models to recognize meaningful patterns in text.

In this project, **spaCy**'s lemmatization capabilities were used to transform reviews into their base forms, enhancing the semantic consistency of the dataset.

The following Python code was used to perform lemmatization on the reviews:

Python Code

```
# Function for lemmatization
def lemmatize_text(text):
    doc = nlp(text)
    return ' '.join([token.lemma_ for token in doc])
# Apply lemmatization to the Review column
df['Review'] = df['Review'].apply(lemmatize_text)
df.head()
```

By applying lemmatization, the dataset becomes more standardized, reducing redundancy caused by word variations and improving the quality of features for model training and rating prediction.

Output:

	Rating	Review	Stopword_Count
0	1	book short not qualify novella guess pay free	15
1	1	not believe good review book like take lesson ...	25
2	1	love book great mystery love story read book a...	12
3	1	like reviewer find book creepy repulse bacchus...	18
4	1	book real let get people turn animal bizarre s...	13

5.21 Choice of Lemmatization Over Stemming

Lemmatization was preferred over stemming for preprocessing textual data in this project because it preserves the semantic meaning of words while reducing them to their base forms. Unlike stemming, which often truncates words arbitrarily (e.g., "running" → "run" but "better" → "bett"), lemmatization considers the context and part-of-speech of words, producing linguistically correct lemmas (e.g., "better" → "good"). This is particularly important for customer reviews, where accurate representation of sentiment and meaning is crucial for rating prediction. By using lemmatization, the project ensures that feature extraction reflects true semantic relationships in the text, leading to better performance of machine learning models.

Comparison of Lemmatization and Stemming

Feature	Lemmatization	Stemming
Definition	Reduces words to their base or dictionary form (lemma)	Reduces words to their root form by chopping off suffixes/prefixes
Preserves Meaning	Yes, context-aware and linguistically accurate	No, may produce non-words or incorrect roots
Example	"running" → "run", "better" → "good"	"running" → "run", "better" → "bett"
Accuracy	High, retains semantic meaning	Lower, may distort meaning
Computational Cost	Slightly higher due to NLP model requirements	Lower, simpler rule-based approach
Use Case	Sentiment analysis, text classification, and NLP tasks requiring semantic understanding	Quick preprocessing, search indexing, or when meaning preservation is not critical

Table 2: Comparison between Lemmatization and Stemming

5.22 Filter out reviews with: Fewer than minimum words and Excessively long text

To maintain consistency and ensure meaningful input for machine learning models, reviews were filtered based on their word count. Very short reviews may lack sufficient context, while excessively long reviews can introduce noise or computational overhead. In this project, reviews with fewer than 3 words or more than 300 words were removed, ensuring that the dataset contains reviews of reasonable length for feature extraction and model training.

The following Python code was used to filter reviews by word count:

Python Code

```
# Define thresholds
min_words = 3 # Minimum words per review
max_words = 300 # Maximum words per review
# Count words in each review
df['Word_Count'] = df['Review'].apply(lambda x: len(x.split()))
# Filter reviews
```

```
df = df[(df['Word_Count'] >= min_words) & (df['Word_Count'] <=
    max_words)].reset_index(drop=True)
print("Dataset shape after filtering:", df.shape)
df[['Review', 'Word_Count']].head()
```

By filtering reviews based on word count, the dataset becomes more standardized, eliminating extremely short or long reviews that could skew feature representation and adversely affect the performance of machine learning models.

Output:

```
Dataset shape after filtering: (36686, 3)
```

	Review	Word_Count
0	way nice story middleage sexually liberate att...	20
1	good setting read take 5 page hop bed nice sto...	30
2	want die twist usual jumper story author defin...	20
3	short usual liked creativity story like read s...	10
4	wow read review people hate little novella thi...	155

5.23 Calculating Word Count

To facilitate analysis of review length and support preprocessing decisions, a `Word_Count` column was added to the dataset. This column records the number of words in each review, providing a quantitative measure of review length. Such information is useful for filtering extreme cases, analyzing textual complexity, and understanding the distribution of review content across the dataset.

The following Python code was used to compute and add the word count:

Python Code

```
# Add a column with word count for each review
df['Word_Count'] = df['Review'].apply(lambda x: len(x.split()))
df[['Review', 'Word_Count']].head()
```

By including the word count as a feature, the preprocessing pipeline gains a simple yet informative metric that can guide further data cleaning, exploration, and feature engineering.

Output:

	Review	Word_Count
0	way nice story middleage sexually liberate att...	20
1	good setting read take 5 page hop bed nice sto...	30
2	want die twist usual jumper story author defin...	20
3	short usual liked creativity story like read s...	10
4	wow read review people hate little novella thi...	155

6. Sample Reviews per Rating

To better understand the textual content and sentiment patterns associated with each rating, a random sample of reviews was extracted for each rating category. Examining sample reviews helps verify the quality of preprocessing, provides qualitative insight into the dataset, and ensures that each rating class contains representative textual examples.

The following Python code was used to sample reviews per rating:

Python Code

```
# Sample reviews from each rating category
for rating, group in df.groupby('Rating'):
   print(f"\n--- Rating: {rating} ---\n")
   sample_reviews = group['Review'].sample(
      n=min(5, len(group)),
      random_state=42
   )
   for i, review in enumerate(sample_reviews, 1):
      print(f"{i}. {review}\n")
```

By inspecting representative reviews from each rating level, researchers can qualitatively confirm that preprocessing steps such as cleaning, lemmatization, and stopword removal have preserved meaningful content. This step also aids in understanding linguistic patterns that may influence rating prediction.

Sample Reviews

```
--- Rating: 1 ---

1. story predictable light reading like harlequin different heroin not
   like
2. book ok author little wordy give way description not bad interested
   flipping filler mary sue heroine titmouse literally tstl not want
   spoil phone sex man ask trust go waysaside stupid thing go
   relationship hero way blackmail heroine like s poopyhead pbut sum
   book overwrite main character defy belief recommend
3. write teenager phrase arbitrary capitalization mean ugh good story
   past second chapter recommend
```

4. book premise shaky charachter force book read read
5. delete book library not frustrate reading book masochism

--- Rating: 2 ---

1. second story ve read kevay good night terrify erotic story ve read well greatrose drive away life terrible boyfriend pick hitchhiker heath immediately attract talk ex couple hour turnon right lady decide check hotel night take far terrible awesome night hotel kind sketchy story skip stomachchurne detail good night traumatic include detail not need like assume clean vibrator different orifice explicitly tell fairly standard erotic fictionthe end bit silly fairy tale fluff
2. sneak peek truly sample story start entice reader continue remind unfortunate lemony snicket series story niece read 12 bonus card small font decipher definitely ya series
3. look short midlife romance explicit sex christmas theme fit billi personally find dull difficult reading character plot predictable not author invest story
4. cute idea m sure love story character find want skip ahead d miss skip long poem not feel like miss alloverall find story bore
5. give story star not flesh bone little background lot hole story

--- Rating: 3 ---

1. good story enjoy way time enjoy fun haunt book
2. book end tell mattheus cindy person new case start book start get phone new case book edit
3. well long time exploit sexual relationship full extreme want like book magic theory supernatural occurrence book romance steampunk set fully develop like steampunk romance book
4. keep pace take age old cougar scenario give bit new life inside look reader different entirely new light 29 hard envision cougar mean 18 yearold let honest find weird 18 desire try relive glory day guy not think 18 guy mature turn head smoking hot s 18okunless werewolf twilightin casecall meregina early 40s josh mid 20 realm possibility day sure stigma attach old woman date young guy find day year okay old man young woman cougar risen yay s brain say deeply ingrained southern heritage family say hmmmmmm think book man 30 woman late 40 s bit different josh college pretty throw bit feel old soul incredibly mature age definite concern not real world not start career think age show lot instance yes overcome not feel real shouldthere disconnect book start strong love character not agree book went feel bit not finger exactly inability relate age experience find yell regina obvious not interested lou boss romantically tempt appear easy way marry sake age have family understand degree time say not interested mull possibility not mull lou think dead husband patrick feel like ton baggage think combine distance biti tell work thoughthe sex orgasm wow sexy scenes regina virile middleage lady guess s woman hit peak maybe sense man hit peak 20 think god idea cruel joke right specie hit mating high year

apart humpffffnear end come book way enjoy end like year jump future like way closure give sweetness dee give 180 early point bookkeepe pace reevaluate thinking cougar give ability experience feeling inside instead outsideri keep pace dee carney 350 star
5. receive short story free storycartel return reviewthe memory light engage short story woman amnesiac live riffraff victorian london weiland manage hold reader interest solid description action pack scene continuous leitmotif light fear haunt amnesiac protagonist writing pacing solid especially like way protagonist loss memory portray keep read mystery woman origin revealedthere noticeable glitch dialogue spoil realistic evocation period victorian riffraff simply not 34he fund big operations34 34e wuz someve big jobs34 34 sort unionize group34 probably right union legalize mid19th century previous call 34societies34 34guilds34however despite generally wellpace write engage story ebook 3 star face story lack depth character 34typical34 not special cause reread story not 34suffer34 reading feel time well spend not obligate provide review

--- Rating: 4 ---

1. enjoy series popular alpha male feisty human female character divide hater good theme human divide readily like outcast hero rage ancient greek capable human female new character date mid 19th century throw peril space making good melodramai find difference character series not great job create complete world law physics stand future
2. thoughly enjoy book ordain street anoite street save know god throw hard choice circumstancesall deviant cross path bring drama suspense story line book captivate believe
3. different spin normal zombie story different spin cause infection blaine return
4. bet not know book tell sturak entertainment youtubeyes link end book watch tablet tablet like know youtube will not work wellthe link review way story tell traditional format begin bit confused far go sincemost book read story take place time read feel like experience themnot story tell playbyplay past adventure like friend tell date go nighti think cute yes feel character endif account thing enjoy bookvideo not let bad review shape opinion story book offer bit page
5. enjoy book love bumble 34phone kitten34 relate predicament get tenacious little chick trouble high heel love keep laugh scary moment recommend

--- Rating: 5 ---

1. exciting fastpaced book not believable character will not lie thrilling book ll read
2. holiday romance great like new wavescrooge twist sista flavor nice writing love
3. love author love book
4. enjoy start series focus micheal eva lay groundwork big battle end

```
series unmate fallen find mate worried end thankfully turn want love
 concept angel book ve set potential oppressor human race death
human race fall not stop human brother m hope odd favour maybe
nonfallen angel oppression destruction th human race fall realise m
look forward locate nephilim hopefully danger rule angel state
fallen continue locate ve mate protect join fight s assume lot
nephilim
```

5. excellent book good story line leave want continue read find happen nexta read story space

7. Data Visualization

7.1 Bar Plot: Review Count per Rating

To better understand the distribution of review ratings in the imbalanced dataset, a count plot was generated. This visualization highlights the unequal representation of each rating category, which is a common characteristic of real-world review data.

The count plot provides insights into the number of reviews per rating, helping to identify potential challenges for machine learning models when training on imbalanced data.

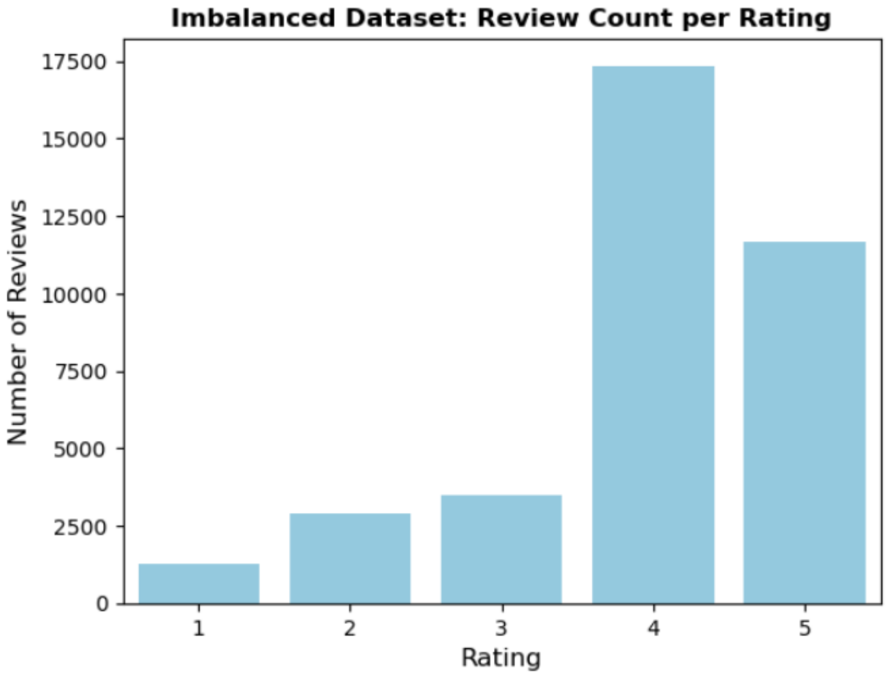
The following Python code was used:

Python Code

```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='Rating', data=df, color='skyblue')
plt.title('Imbalanced Dataset: Review Count per Rating', fontsize=12,
    weight='bold')
plt.xlabel('Rating', fontsize=12)
plt.ylabel('Number of Reviews', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

Key Observations:

- Higher ratings typically dominate the dataset, while lower ratings are less frequent.
- This imbalance can lead to model bias, where predictions favor the majority classes.
- Visualizing the distribution aids in deciding whether to apply resampling, class weighting, or specialized evaluation metrics.



Imbalanced Dataset Review Count per Rating Visualization

7.2 Pie Chart: Rating Distribution

To provide a clear visual summary of the class imbalance, a pie chart was generated showing the proportion of reviews for each rating category. Pie charts are effective for highlighting the relative size of each class, helping to identify dominant and underrepresented ratings.

The Python code used:

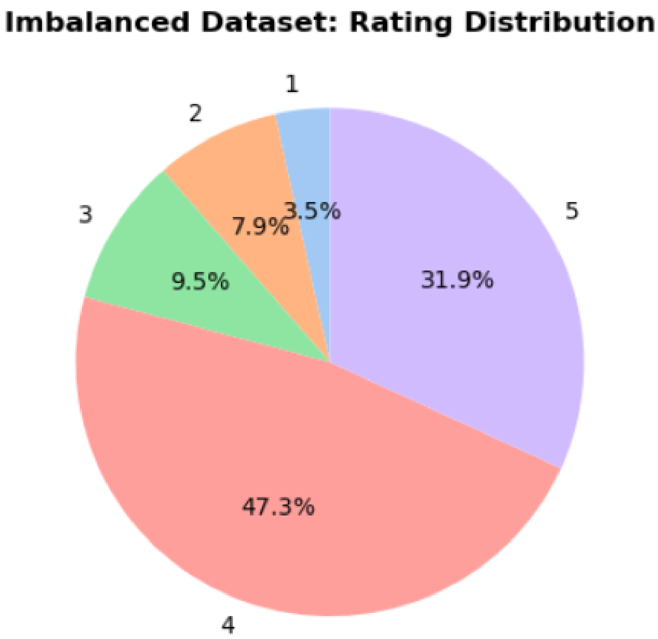
Python Code

```
import matplotlib.pyplot as plt
import seaborn as sns
# Count of reviews per rating
rating_counts = df['Rating'].value_counts().sort_index()
# Labels and values for pie chart
categories = rating_counts.index
values = rating_counts.values
plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=90,
    colors=sns.color_palette("pastel"))
plt.title("Imbalanced Dataset: Rating Distribution", fontsize=12,
    weight='bold')
plt.show()
```

Key Observations:

- The pie chart clearly shows that higher ratings dominate, while lower ratings are underrepresented.
- This visualization complements the count plot, providing a percentage-based view of the class distribution.

- Understanding this imbalance is critical for model selection, performance evaluation, and potential resampling strategies.



Imbalanced Dataset Rating Distribution Visualization

7.3 Histogram: Word Count Distribution

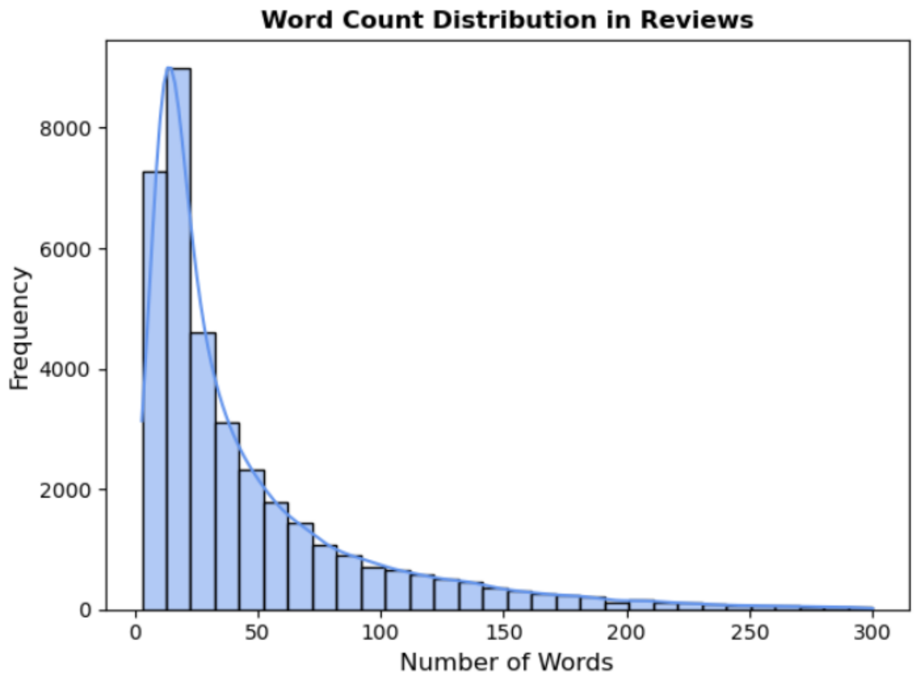
Analyzing the distribution of review lengths provides insights into the textual characteristics of the dataset. A histogram was created to visualize the word count across all reviews, highlighting the frequency of reviews with different lengths. This analysis helps identify extreme cases and ensures that the dataset contains reviews of reasonable length for feature extraction and model training.

The following Python code was used to generate the histogram:

Python Code

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(df['Word_Count'], bins=30, kde=True, color='cornflowerblue
    ')
plt.title('Word Count Distribution in Reviews', fontsize=12, weight='
    bold')
plt.xlabel('Number of Words', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.tight_layout()
plt.show()
```

This visualization confirms that the majority of reviews fall within a moderate word count range, validating the earlier filtering step and ensuring that the dataset is suitable for text-based machine learning models.



Imbalanced Dataset Word Count Distribution in Review Visualization

7.4 Box Plot: Word Count Distribution by Rating

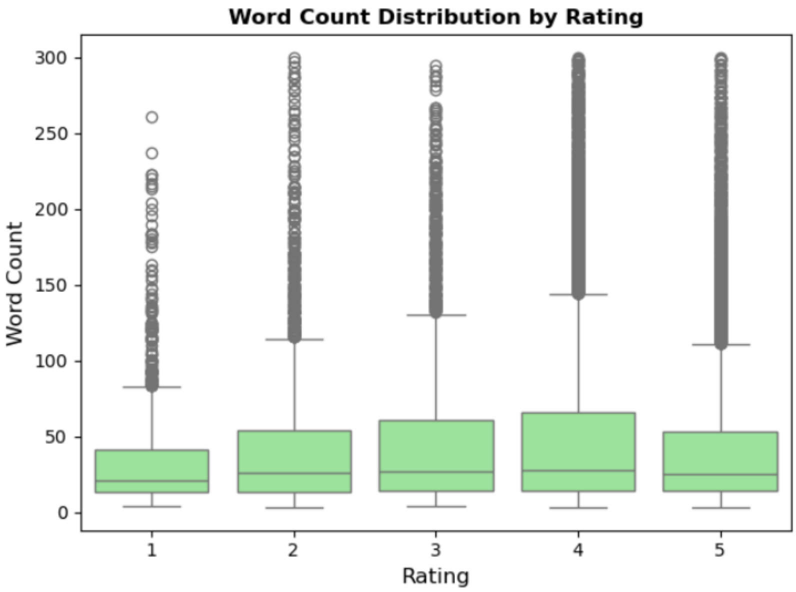
To examine how review length varies across different rating levels, a box plot was created showing the distribution of word counts for each rating category. This visualization helps identify patterns, such as whether higher or lower ratings tend to have longer or shorter reviews, and can inform feature engineering or preprocessing decisions.

The following Python code was used to generate the box plot:

Python Code

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.boxplot(x='Rating', y='Word_Count', data=df, color='lightgreen')
plt.title('Word Count Distribution by Rating', fontsize=12, weight='
    bold')
plt.xlabel('Rating', fontsize=12)
plt.ylabel('Word Count', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()
```

This visualization provides insights into the relationship between review length and ratings, helping ensure that text preprocessing and modeling decisions account for differences in review characteristics across rating categories.



Word Count Distribution by Rating Visualization

8. Natural Language Processing (NLP)

8.1 Shuffling the Dataset

Shuffling the dataset is an important preprocessing step to ensure that the order of reviews does not introduce any unintended bias during model training. Randomizing the sequence of data helps machine learning algorithms learn patterns more effectively and prevents overfitting to any particular sequence of ratings.

The following Python code was used to shuffle the dataset:

Python Code

```
# Shuffle the dataset
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
df.head()
```

By shuffling the dataset, the training and evaluation processes become more robust, ensuring that the model is exposed to a representative mix of all rating categories throughout the learning process.

	Review	Rating
0	book make wish young explore sexuality college...	4
1	dull moment get catch page rachel friend danny...	4
2	movie man s sleep discover s secret undergroun...	5
3	joann carter fill lovely story vivid descripti...	4
4	short novella quick read story interesting ins...	4

8.2 Train-Test Split

To evaluate the performance of machine learning models, the dataset was divided into training and testing sets. The training set is used to learn patterns from the data, while the testing set evaluates how well the model generalizes to unseen reviews. An 90:10 split was applied, with stratification based on ratings to ensure that all rating categories are proportionally represented in both sets.

The following Python code was used to split the dataset:

Python Code

```
from sklearn.model_selection import train_test_split
# Features and target
X = df['Review']
y = df['Rating']
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.1, random_state=42, stratify=y
)
print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

By stratifying the split, the distribution of ratings in the training and testing sets mirrors that of the overall dataset. This ensures balanced evaluation and reliable performance metrics across all rating classes.

```
Training set shape: (29348,)
Testing set shape: (7338,)
```

8.3 Stratified Split

A stratified split is a method of dividing a dataset into training and testing sets while preserving the proportion of each class in both sets. In classification tasks, especially with imbalanced classes, a simple random split may result in uneven representation of some categories, which can bias the model. Stratification ensures that the distribution of target labels (e.g., review ratings 1–5) in the train and test sets matches the distribution in the original dataset.

Example:

Original dataset: 50% rating 5, 20% rating 4, 30% other ratings

Stratified split (90:10) $\rightarrow$ Training set and testing set will maintain the same 50:20:30 ratio.

This technique helps improve model evaluation reliability and prevents certain classes from being underrepresented in either the training or testing set.

8.4 TF-IDF Vectorization

After preprocessing the text, the next step is converting textual data into numerical features that machine learning models can understand. TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used technique for this purpose. It assigns weights to each term in the corpus based on its importance in a document relative to the entire dataset.

TF-IDF Concept

TF-IDF combines two metrics: Term Frequency (TF) and Inverse Document Frequency (IDF).

- **Term Frequency (TF):** Measures how frequently a word occurs in a document.

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Inverse Document Frequency (IDF):** Measures how important a term is by reducing the weight of terms that appear in many documents (common words) and increasing the weight for rarer terms.

$$IDF(t) = \log \frac{N}{1+n_t}$$

  Where:

  - N = Total number of documents
  - n_t = Number of documents containing term t

- **TF-IDF Weight:** Combines TF and IDF to assign a weight to each term.

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

8.5 TF-IDF Implementation

To convert textual reviews into numerical features suitable for machine learning models, Term Frequency-Inverse Document Frequency (TF-IDF) vectorization was applied. TF-IDF assigns higher weight to words that are important in a particular document but less frequent across the corpus, allowing the model to focus on meaningful terms. In this project, word-level TF-IDF was computed using unigrams, bigrams, and trigrams to capture both individual words and short phrases.

Parameters used:

- `sublinear_tf=True`: Applies logarithmic scaling to term frequency.
- `max_features=5000`: Limits the feature space to the 5000 most informative terms.
- `ngram_range=(1,3)`: Includes unigrams, bigrams, and trigrams.
- `min_df=3` and `max_df=0.9`: Filters out extremely rare and overly common terms.

Python Code

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Word-level TF-IDF
vectorizer = TfidfVectorizer(
   sublinear_tf=True,
   max_features=5000,
   ngram_range=(1, 3),
   min_df=3,
   max_df=0.9
)
```

```
# Fit TF-IDF on training data and transform
X_train_tfidf = vectorizer.fit_transform(X_train)
# Transform test data
X_test_tfidf = vectorizer.transform(X_test)
print("Training TF-IDF shape:", X_train_tfidf.shape)
print("Testing TF-IDF shape:", X_test_tfidf.shape)
```

By applying TF-IDF vectorization, textual reviews are transformed into a numerical feature matrix, enabling machine learning models to learn patterns and relationships between words/phrases and review ratings efficiently.

Output:

```
Training TF-IDF shape: (29348, 5000)
Testing TF-IDF shape: (7338, 5000)
```

8.6 Choice of TF-IDF Over Bag of Words (BoW)

In this project, TF-IDF was preferred over the traditional Bag of Words (BoW) approach for feature extraction due to its ability to emphasize informative words while reducing the impact of commonly occurring, less meaningful terms. While BoW simply counts the frequency of words in a document, TF-IDF scales these counts by the inverse document frequency, assigning lower weight to words that appear across many reviews (e.g., “product”, “good”) and higher weight to distinctive words that provide meaningful sentiment cues.

This weighting mechanism allows machine learning models to focus on terms that are more predictive of review ratings, improving model performance, robustness, and interpretability. TF-IDF is particularly effective in text classification tasks like review rating prediction, where distinguishing between subtle differences in word usage can significantly influence model accuracy.

8.7 Comparison of TF-IDF and Bag of Words (BoW)

Feature	**TF-IDF**	**Bag of Words (BoW)**
Definition	Weights words by frequency & uniqueness	Counts word occurrences
Word Importance	Highlights rare/informative words	All words treated equally
Context	Considers significance in corpus	Ignores importance/context
Scaling	Inverse document frequency applied	Raw counts only
Use Case	Sentiment analysis, text classification	Basic text analysis
Performance	Often higher accuracy & interpretability	May lower model accuracy

Table 3: Comparison of TF-IDF and Bag of Words (BoW) for feature extraction

8.8 Converting TF-IDF Matrices to DataFrames

After computing TF-IDF features, the resulting matrices are sparse numeric arrays. Converting them into pandas DataFrames allows for easier inspection, analysis, and integration with machine learning pipelines. Each column in the DataFrame corresponds to a feature (word, bigram, or trigram), and each row represents a review. This structured format facilitates visualization, feature selection, and debugging during model development.

Python Code

```
# Get feature names from the vectorizer
feature_names = vectorizer.get_feature_names_out()
# Convert TF-IDF matrices to DataFrames
X_train_df = pd.DataFrame(X_train_tfidf.toarray(), columns=
    feature_names)
X_test_df = pd.DataFrame(X_test_tfidf.toarray(), columns=feature_names)
print("\nTrain TF-IDF DataFrame shape:", X_train_df.shape)
print("Test TF-IDF DataFrame shape:", X_test_df.shape)
X_train_df.head()
X_train_df.tail()
```

By converting TF-IDF matrices to DataFrames, researchers can easily explore features, identify top-weighted terms, and ensure that the preprocessing pipeline correctly represents the textual data in numerical form for machine learning models.

Output:

```
Train TF-IDF DataFrame shape: (29348, 5000)
Test TF-IDF DataFrame shape: (7338, 5000)
```

	099	10	10 year	100	11	12	12 star	13	14	15	...	zach	zack	zane	zero	zoe	zombie	zombie book	zombie story	zone	zu
0	0.0	0.149279	0.181647	0.0	0.0	0.0	0.0	0.287235	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5000 columns

	099	10	10 year	100	11	12	12 star	13	14	15	...	zach	zack	zane	zero	zoe	zombie	zombie book	zombie story	zone	zu
29343	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29344	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29345	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29346	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29347	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5000 columns

8.9 Saving the TF-IDF Vectorizer

To enable consistent feature extraction for future use and deployment, the trained TF-IDF vectorizer was saved to disk. Persisting the vectorizer ensures that the same vocabulary and feature mappings are applied to new data, maintaining consistency in model predictions. This is especially important when deploying machine learning models in production environments.

Python Code

```
import os
import joblib
# Create directory to save models
model_dir = r"..\models"
os.makedirs(model_dir, exist_ok=True)
# Save word-level TF-IDF vectorizer
```

```
word_vectorizer_path = os.path.join(model_dir, "tfidf_balanced.pkl")
joblib.dump(vectorizer, word_vectorizer_path)
print(f"TF-IDF Vectorizer saved at: {word_vectorizer_path}")
```

By saving the TF-IDF vectorizer, the preprocessing pipeline becomes reproducible and allows new reviews to be transformed consistently before feeding them into trained models.

9. Machine Learning Model Training and Evaluation

9.1 Overview

Predicting review ratings from textual feedback was formulated as a multi-class classification problem. Several supervised learning algorithms were explored to determine the most effective model for this task. Textual features were extracted using TF-IDF vectorization, capturing the significance of unigrams, bigrams, and trigrams. All experiments were conducted on a balanced dataset, ensuring equal representation of each rating category to prevent bias in model predictions and promote reliable evaluation.

9.2 Models Evaluated

The following machine learning models were assessed for their suitability in review rating prediction:

- **Logistic Regression (LR):** A linear model for multi-class classification, effective for linearly separable data.
- **Support Vector Machine (LinearSVC):** Utilizes a linear kernel to separate classes with maximum margin.
- **Multinomial Naive Bayes (MultinomialNB):** A probabilistic model well-suited for text-based data and word count features.
- **Decision Tree (DT):** A tree-based classifier that captures non-linear relationships in the feature space.
- **Random Forest (RF):** An ensemble of decision trees that leverages bagging and feature randomness to improve generalization.

9.3 Model Selection – Logistic Regression

After evaluating performance across multiple algorithms, Logistic Regression was selected as the primary model due to its high accuracy, interpretability, and efficiency on TF-IDF features derived from textual reviews. Logistic Regression provided consistent predictions across all rating categories and demonstrated robust generalization on the test dataset.

Logistic Regression on Imbalanced Dataset

To assess the model's robustness under **class imbalance**, Logistic Regression was trained and evaluated on the imbalanced dataset. Handling imbalanced data requires careful consideration, such as using **class weights**, to prevent bias toward majority classes.

Implementation Details

- **Model Initialization:**
  - `max_iter=6000` ensures convergence on large sparse data.
  - `solver='saga'` efficiently handles high-dimensional TF-IDF features.
  - `multi_class='multinomial'` supports multi-class classification.
  - `class_weight` adjusts for class imbalance to improve minority class prediction.
- **Training and Prediction:**
  - The model was trained on TF-IDF features and used to predict ratings for the test set.
- **Evaluation Metrics:**
  - **Accuracy:** Overall correctness of predictions.
  - **Classification Report:** Includes precision, recall, and F1-score for each rating class.
  - **Confusion Matrix:** Visualizes correct and misclassified predictions across all ratings.

Python Code

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
   confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

lr_model = LogisticRegression(
   max_iter=6000,
   solver='saga',
   multi_class='multinomial',
   class_weight=class_weights_dict,
   random_state=42
)

lr_model.fit(X_train_tfidf, y_train)
y_pred = lr_model.predict(X_test_tfidf)

acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, digits=4))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
   classes)
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix - Logistic Regression (Imbalanced Data)")
plt.show()
```

Model Training Logic

The Automated Review Rating System employs Logistic Regression for multi-class classification of review ratings. The training pipeline follows these steps:

1. **Feature Extraction:**
    - Textual reviews are transformed into TF-IDF features, capturing the importance of unigrams, bigrams, and trigrams.
    - This converts raw text into a numerical feature matrix suitable for machine learning models.
2. **Data Splitting:**
    - The dataset is split into training and testing sets using a stratified split to ensure proportional representation of each rating category.
3. **Model Initialization:**
    - Logistic Regression is configured with `saga` solver, `multinomial` multi-class setting, and `class_weight` to address potential class imbalance.
    - High iteration count (`max_iter=5000-6000`) ensures convergence for sparse TF-IDF matrices.
4. **Training:**
    - The model learns weight coefficients for each TF-IDF feature, optimizing the likelihood of correctly predicting review ratings.
5. **Prediction:**
    - The trained model predicts ratings for the test set, producing a set of predicted labels for evaluation.

Evaluation Results and Interpretation

The Logistic Regression model was evaluated on the imbalanced test dataset, achieving an overall accuracy of 0.5213. Detailed performance metrics for each rating category were obtained using a classification report and confusion matrix.

Performance Across Ratings

- **Higher ratings (4 and 5):** Demonstrate better precision and F1-scores due to their larger representation in the dataset.
- **Lower ratings (1, 2, 3):** Show lower precision but moderate recall, indicating that the model tends to over-predict higher ratings for minority classes.

Macro vs. Weighted Average

- **Macro Average (unweighted) F1-score:** 0.4255, reflecting performance without considering class imbalance.

- **Weighted Average F1-score:** 0.5317, accounting for the number of instances per class and highlighting the model's strength on majority ratings.

Insights from Confusion Matrix

- Misclassifications frequently occur between adjacent rating levels, e.g., 4-star reviews predicted as 5-star.
- This pattern is consistent with the subjective nature of reviews, where sentiment can lie between categories.

Evaluation Metrics

- **Accuracy:** Proportion of correctly predicted ratings over total test samples. High accuracy indicates effective learning from TF-IDF features.
- **Precision:** Fraction of correct predictions among all predictions for a class.
- **Recall:** Fraction of correctly predicted instances among all actual instances of a class.
- **F1-Score:** Harmonic mean of precision and recall, balancing both metrics.

Interpretation

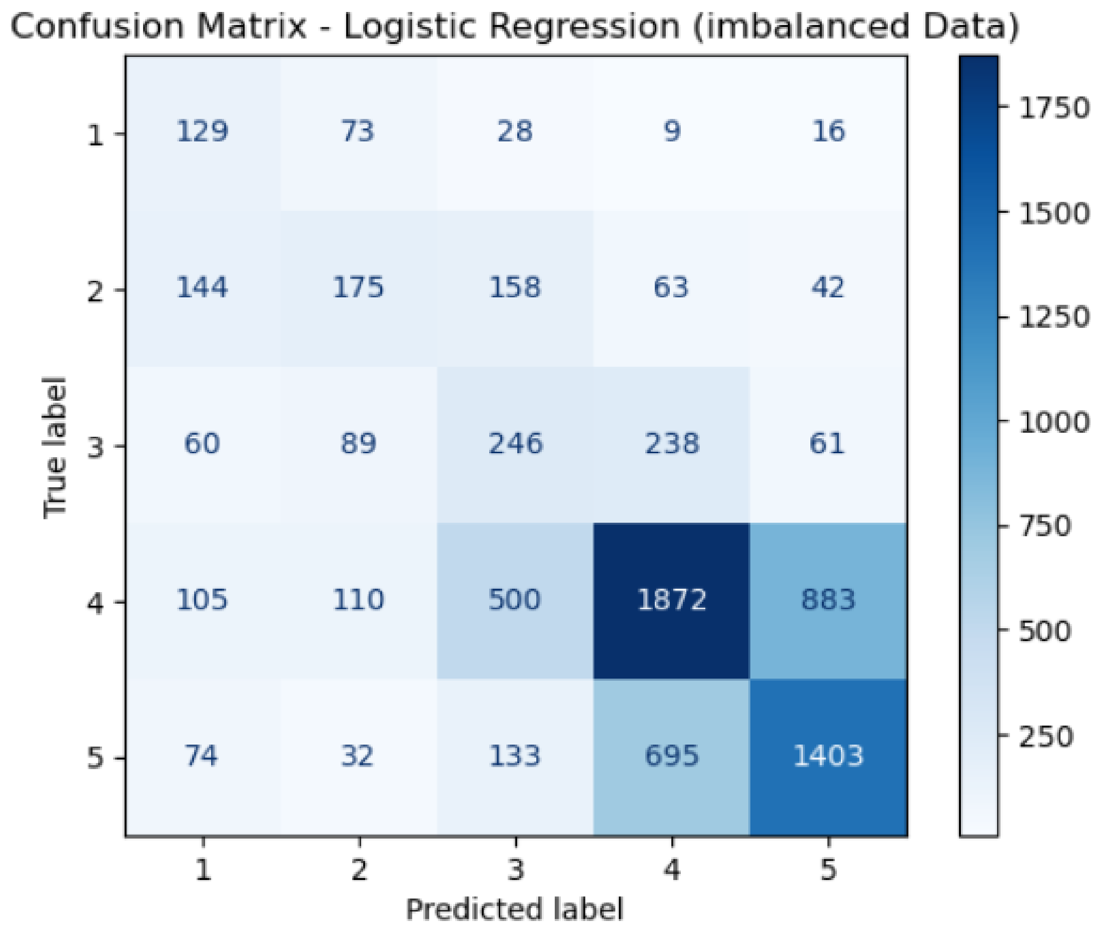
- The model captures meaningful textual patterns, with stronger performance on well-represented ratings and slightly lower performance on underrepresented classes.
- Class weighting improves recall for minority classes but cannot fully compensate for severe imbalance.
- Visualizations like confusion matrices and pie/count plots aid in identifying model weaknesses and guide further improvements, such as resampling techniques or hyperparameter tuning.

```
Accuracy: 0.5213

Classification Report:
              precision    recall  f1-score   support

           1     0.2520    0.5059    0.3364       255
           2     0.3653    0.3007    0.3299       582
           3     0.2310    0.3545    0.2797       694
           4     0.6507    0.5395    0.5899      3470
           5     0.5834    0.6003    0.5917      2337

    accuracy                         0.5213      7338
   macro avg     0.4165    0.4602    0.4255      7338
weighted avg     0.5531    0.5213    0.5317      7338
```



Conclusion

The evaluation demonstrates that while Logistic Regression performs robustly on imbalanced datasets, careful handling of dataset distribution, feature engineering, and evaluation metrics is essential for real-world review rating prediction.

Saving the Trained Model

After training and evaluating the Logistic Regression model on the imbalanced dataset, the model was persisted to disk for future use. Saving the model allows reproducibility, deployment, and avoids retraining every time predictions are needed.

Implementation Details

- The trained model was saved using `joblib`, which efficiently serializes large machine learning models and sparse matrices.
- A dedicated folder `models/` was created to organize all model artifacts.
- The saved model file can be loaded later for prediction or evaluation without retraining.

Python Code Used

```
import joblib
import os
model_dir = r"..\models"
os.makedirs(model_dir, exist_ok=True)
model_path = os.path.join(model_dir, 'Model_Imbalanced.pkl')
# Save the trained Logistic Regression model
joblib.dump(lr_model, model_path)
print(f"Model saved successfully at {model_path}")
```

10. Conclusion

Conversely, training on an imbalanced dataset favored majority classes, resulting in higher accuracy for well-represented ratings (4 and 5 stars) but significantly lower performance for minority classes. Precision and recall metrics indicate that the model often misclassifies underrepresented ratings, reflecting the inherent bias introduced by imbalanced training. While the overall accuracy may appear satisfactory on datasets that reflect similar class distributions, the imbalanced model lacks fairness across all categories, limiting its applicability in scenarios where equitable prediction is required. These observations highlight the critical trade-off between overall accuracy and class-wise fairness in multi-class review rating prediction.

11. Documentation Link

- **CROSS-TESTING: Balanced vs Imbalanced Models Documentation** Documentation Link
- **Frontend: Automated Review Rating System Documentation** Documentation Link