FINAL PROJECT : SENTIMENT ANALYSIS AND RECOMMENDATION

Lakshman Raaj Senthil Nathan
002737381
Northeastern University
Boston, MA
senthilnathan.l@northeastern.edu

Sivaranjani Suresh Manivannan
002742197
Northeastern University
Boston, MA
s.si@northeastern.edu

Abstract

In this project, sentiment analysis was performed on a dataset sourced from Amazon Books Reviews - Goodreads-books reviews and descriptions of each book. The dataset comprises feedback from 3 million users on 212,404 unique books. This data is a subset of the Amazon review Dataset, encompassing product reviews and metadata from Amazon, spanning the period from May 1996 to July 2014. The primary objective was to categorize the sentiment of reviews into 'negative', 'neutral', and 'positive' based on the 'review/score' column.

Datasets

We utilized the "Amazon Books Reviews" dataset for this project. The dataset contains two csv files namely ‘Books_rating.csv’, ‘books_data.csv’ having the following attributes:

1. Books_rating.csv

S.no	Columns	Rows	Datatype	Non-null values
1	Id	3.00m	Object	3.00m
2	Title	3.00m	Object	3.00m
3	Price	3.00m	Float	3.00m
4	User_id	3.00m	Object	3.00m
5	profileName	3.00m	Object	3.00m
6	review/helpfulness	3.00m	Object	3.00m
7	review/score	3.00m	Float	3.00m
8	review/time	3.00m	Object	3.00m
9	review/summary	3.00m	Object	3.00m
10	review/text	3.00m	Object	3.00m

- **id:** The unique identifier of the book.
- **Title:** The title of the book.
- **Price:** The price of the book.
- **User_id:** The unique identifier of the user who rated the book.
- **profileName:** The name of the user who rated the book.
- **review/helpfulness:** Helpfulness rating of the review (e.g., 2/3).
- **review/score:** Rating ranging from 0 to 5 for the book.
- **review/time:** Time when the review was given.
- **review/summary:** The summary of a text review.
- **review/text:** The full text of the review.

The 'review/score' column was used to determine the sentiment labels: 'negative', 'neutral', and 'positive' for the corresponding reviews in the 'review/text' column.

2. books_data.csv

S.no	Columns	Rows	Datatype	Non-null values
1	Title	212,404	Object	212,403
2	description	212,404	Object	143,962
3	authors	212,404	Float	180,991
4	Image	212,404	Object	160,329
5	previewLink	212,404	Object	188,568
6	publisher	212,404	Object	136,518
7	publishedDate	212,404	Float	187,099
8	infoLink	212,404	Object	188,568
9	categories	212,404	Object	171,205
10	ratingsCount	212,404	Object	49,752

- **Title:** The title of the book.
- **Description:** A brief summary or overview of the book's content and storyline.
- **Authors:** The names of the authors who wrote the book.
- **Image:** The URL linking to the book cover image.
- **PreviewLink:** The link that allows access to a preview or more detailed information about the book on Google Books.

- **Publisher:** The name of the publishing company responsible for releasing the book.
- **PublishedDate:** The date when the book was officially published.
- **InfoLink:** The link that provides additional information about the book on Google Books.
- **Categories:** The genres or categories to which the book belongs.
- **RatingsCount:** The total count of ratings received for the book, indicating its popularity or reader feedback.

Data Characteristics

The dataset consists of feedback from a vast user base on a diverse range of books. With 212,404 unique books and 3 million user reviews, the dataset provides a rich and varied source of information. The 'review/score' column, ranging from 0 to 5, serves as the basis for sentiment categorization. The reviews cover a wide span of time, offering insights into user sentiments over two decades of book reviewing on Amazon. The diversity in book genres, user profiles, and review lengths adds complexity and depth to the analysis of sentiment in this dataset.

Why Are These Interesting Datasets?

The "Amazon Books Reviews - Goodreads-books reviews and descriptions of each book" dataset presents a compelling opportunity for sentiment analysis due to its scale, diversity, and historical context.

1. **Scale:** With feedback from 3 million users on 212,404 unique books, the dataset offers a vast collection of reviews, reflecting a wide array of reader preferences and opinions. Analyzing such a large-scale dataset allows for robust insights into the patterns of user sentiment across different books and genres.
2. **Diversity:** The dataset encompasses books from various genres, catering to different interests and demographics. This diversity ensures a heterogeneous mix of reviews, allowing for a nuanced exploration of sentiment across different literary categories. Studying sentiment in diverse contexts provides valuable information for both the publishing industry and readers seeking tailored recommendations.
3. **Historical Context:** Spanning from May 1996 to July 2014, the dataset covers a substantial timeframe, capturing evolving trends in literature and reader preferences over two decades. Analyzing sentiment within this historical context enables the identification of long-term patterns, allowing for a deeper understanding of how readers' sentiments have changed over time and in response to broader societal and cultural shifts.

4. **User Engagement:** The inclusion of attributes such as 'helpfulness rating' and 'review/helpfulness' provides an additional layer of insight into user engagement. Understanding how users perceive the helpfulness of reviews can offer valuable feedback to both authors and readers, influencing purchasing decisions and shaping the online book review ecosystem.

In summary, the dataset's scale, diversity, historical depth, and indicators of user engagement make it an intriguing and valuable resource for conducting sentiment analysis. The findings derived from this dataset can provide significant insights into the factors influencing reader sentiment and contribute to enhancing user experiences in the realm of book reviews and recommendations.

Data Cleaning

The dataset underwent several preprocessing steps to ensure its quality and relevance for sentiment analysis. The following operations were performed:

1. **Column Selection and Renaming:** Unnecessary columns such as 'Id', 'Title', 'Price', 'User_id', 'profileName', 'review/helpfulness', 'review/time', and 'review/summary' were dropped from the dataset. The 'review/text' column was renamed to 'review', and the 'review/score' column was renamed to 'label'.

```
df = df.drop(['Id', 'Title', 'Price', 'User_id', 'profileName',
'review/helpfulness', 'review/time', 'review/summary'], axis=1)

df = df.rename(columns={"review/text": "review", "review/score":
"label"})
```

2. **Handling Missing Values:** Rows containing missing values were removed to ensure the dataset's completeness and reliability.

```
df = df.dropna()
```

3. **Removing Duplicate Entries:** Duplicate entries, if any, were identified and removed to maintain the uniqueness of the dataset.

```
df = df.drop_duplicates()
```

4. **Sentiment Labeling:** Sentiment labels were assigned based on the 'review/score' ratings. Ratings greater than 3.0 were considered positive (labeled as 2), ratings equal to 3.0 were considered neutral (labeled as 1), and ratings less than 3.0 were considered negative (labeled as 0).

```
def mark_sentiment(rating):
        if rating > 3.0:
            return 2
        elif rating == 3.0:
            return 1
        else:
            return 0
```

5. **Visualizing Label Distribution:** A pie chart was generated to visualize the distribution of sentiment labels (positive, neutral, and negative) in the cleaned dataset. This visualization provides an overview of the balanced or imbalanced nature of the dataset.

These cleaning steps ensure that the dataset is well-prepared for subsequent analysis, allowing for accurate and meaningful insights to be derived during the sentiment analysis process.

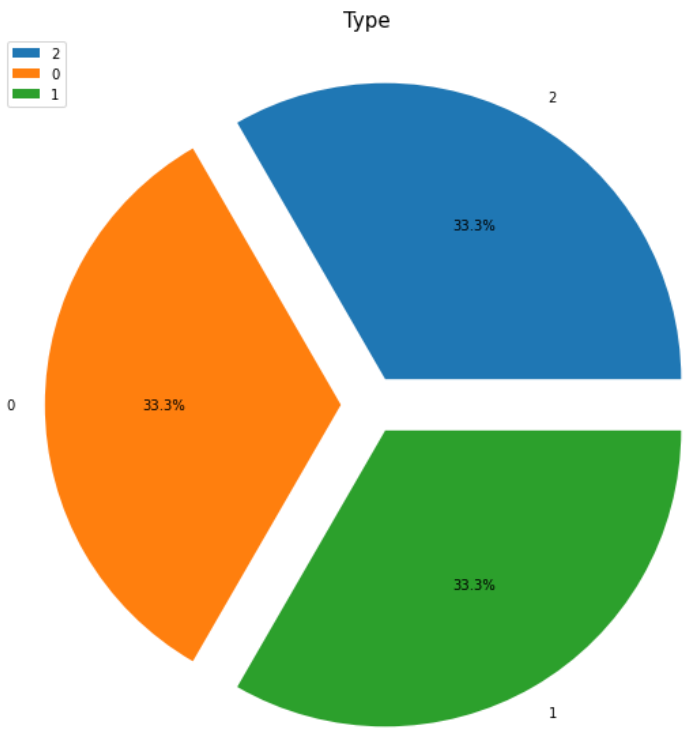
Balancing Classes

In the initial dataset, the distribution of sentiment labels (positive, neutral, and negative) might be imbalanced, potentially leading to biased results during the analysis. To address this, the classes were balanced by limiting the number of samples for each sentiment category.

1. **Class Balancing:** To achieve a balanced representation of sentiments, a subset of the dataset was created by selecting a fixed number of samples from each sentiment class. For instance, 500 samples were randomly chosen for each sentiment category (positive, neutral, and negative). This ensured that each sentiment class had an equal representation in the balanced dataset.

```
# Select a subset of samples from each sentiment class
class_0 = df[df["label"] == 0].iloc[:500]
class_1 = df[df["label"] == 1].iloc[:500]
class_2 = df[df["label"] == 2].iloc[:500]
# Concatenate the balanced samples
df = pd.concat([class_2, class_0, class_1])
```

After balancing the classes, a pie chart was generated to visualize the updated distribution of sentiment labels in the balanced dataset. The pie chart provides a clear illustration of the equal distribution achieved among the three sentiment classes.



2. **Categorizing Sentiments:** To facilitate further analysis, a function was defined to categorize the numerical sentiment labels (2 for positive, 0 for negative, and 1 for neutral) into corresponding textual categories ('positive', 'negative', and 'neutral'). This categorization aids in the interpretation and visualization of sentiment analysis results.

```
def categorize_sentiment(score):
    if score == 2:
        return 'positive'
    elif score == 0:
        return 'negative'
    else:
        return 'neutral'
```

The 'sentiment' column was added to the DataFrame, providing a clear textual representation of the sentiment labels, making it easier to comprehend the sentiment distribution and conduct subsequent analyses. The 'sentiment' column allows for a more intuitive understanding of the sentiment categories assigned to the reviews in the balanced dataset, facilitating insightful analysis and interpretation.

Text Cleaning

Text cleaning is a crucial step in natural language processing (NLP) tasks, ensuring that textual data is standardized and devoid of unnecessary elements. The following operations were performed on the 'review' column to prepare the text data for sentiment analysis:

1. **Lowercasing:** All text was converted to lowercase to maintain uniformity and prevent the model from treating words with different cases as distinct entities. Lowercasing simplifies the analysis by treating words like "Word" and "word" as the same token.

```
df['review'] = df['review'].str.lower()
```

Lowercasing helps avoid redundancy and ensures consistency in the dataset, making subsequent text processing steps more effective.

2. **Removing Special Characters, Punctuation, and Extra Whitespace:** Special characters, punctuation marks, and additional whitespace can introduce noise in the text data. Regular expressions were used to remove these elements, leaving only alphanumeric characters and spaces. This process aids in focusing the analysis on the essential textual content.

```
# Remove special characters, punctuation, and extra whitespace

df['review'] = df['review'].apply(lambda x: re.sub(r'[^a-zA-Z0-
9\s]', '', x))

# Remove extra whitespace

df['review'] = df['review'].apply(lambda x: re.sub(r'\s+', ' ',
x).strip())
```

By eliminating non-alphanumeric characters and extra spaces, the text data becomes more concise and standardized, enhancing the accuracy of subsequent NLP tasks such as tokenization and feature extraction.

Data Augmentation

Data augmentation is a technique commonly used in machine learning to artificially increase the diversity of the training dataset. By introducing variations in the data, the model becomes more robust and capable of handling a wider range of inputs. In the context of natural language processing (NLP), data augmentation methods can be applied to text data to enhance the performance and generalization of sentiment analysis models. In this project, several text augmentation techniques were employed:

1. **Synonym Replacement:** Words in the text were replaced with their synonyms. Synonyms provide similar meanings, and this operation enriches the vocabulary of the dataset, making the model more adaptable to different word choices used by reviewers.

```
def synonym_replacement(text):
    words = nltk.word_tokenize(text)
    new_words = []
    for word in words:
        syns = wordnet.synsets(word)
        if syns:
            new_word = syns[0].lemmas()[0].name()
            if new_word != word:
                new_words.append(new_word)
            else:
                new_words.append(word)
        else:
            new_words.append(word)
    return ' '.join(new_words)
```

2. **Word Shuffling:** The order of words in sentences was randomly shuffled. This technique disrupts the sequential structure of sentences, forcing the model to focus on word meanings rather than word positions.

```
def word_shuffle(text):
    words = nltk.word_tokenize(text)
    random.shuffle(words)
    return ' '.join(words)
```

3. **Adding Noise:** Noise was introduced to the text by randomly modifying individual characters within words. This operation simulates typographical errors or variations in spelling, encouraging the model to be more resilient to noisy or misspelled text.

```
def add_noise(text, noise_level=0.1):
    words = nltk.word_tokenize(text)
    noisy_words = []
    for word in words:
        if random.random() < noise_level:
            noisy_words.append(''.join(random.sample(word, 
len(word))))
        else:
            noisy_words.append(word)
    return ' '.join(noisy_words)
```

These augmentation techniques were applied to the original reviews in the dataset, creating augmented versions of the text data. The augmented samples were combined with the original dataset, expanding the training data with diverse textual variations. This augmented dataset, with increased variability and complexity, enhances the model's ability to generalize unseen data and improves its overall performance in sentiment analysis tasks.

By employing these text augmentation techniques, the dataset's richness and diversity were amplified, ensuring a more comprehensive training experience for the sentiment analysis model. This augmented dataset forms a robust foundation for training machine learning models capable of capturing the subtleties and nuances in user sentiments expressed in online book reviews.

Data Preprocessing

Data preprocessing plays a fundamental role in preparing textual data for analysis. The raw text obtained from various sources often contains noise, irrelevant information, and inconsistencies. Preprocessing steps are essential to ensure the data is clean, uniform, and ready for further analysis. In this project, the following preprocessing steps were performed on the review text:

1. **Removing Stop Words:** Stop words are common words such as "and", "the", "is", etc., which do not carry significant meaning and can be safely removed from the text. Removing stop words helps reduce the dimensionality of the data and focus on the more meaningful words.

```
def clean_text(text):
    # Remove stop words
    stop_words = set(stopwords.words('english'))
    words = text.split()
    words = [word for word in words if word not in stop_words]
    text = ' '.join(words)
    return text
```

2. **Lemmatization:** Lemmatization involves reducing words to their base or root form. This step ensures that different forms of the same word are treated as a single entity, improving the consistency of the textual data. For example, "running," "ran," and "runs" would all be reduced to "run."

```
def clean_text(text):
    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    words = text.split()
    words = [lemmatizer.lemmatize(word) for word in words]
    text = ' '.join(words)
    return text
```

3. **Tokenization:** Tokenization breaks down the text into individual words or tokens. It is a crucial step before text analysis, as it converts continuous text into discrete tokens, allowing for further processing and analysis at the word level.

```
import nltk
from nltk.tokenize import word_tokenize

# Download NLTK resources (only required once)
nltk.download('punkt')

# Tokenize the review text and store it in a new column 'tokens'
df['tokens'] = df['review'].apply(word_tokenize)
```

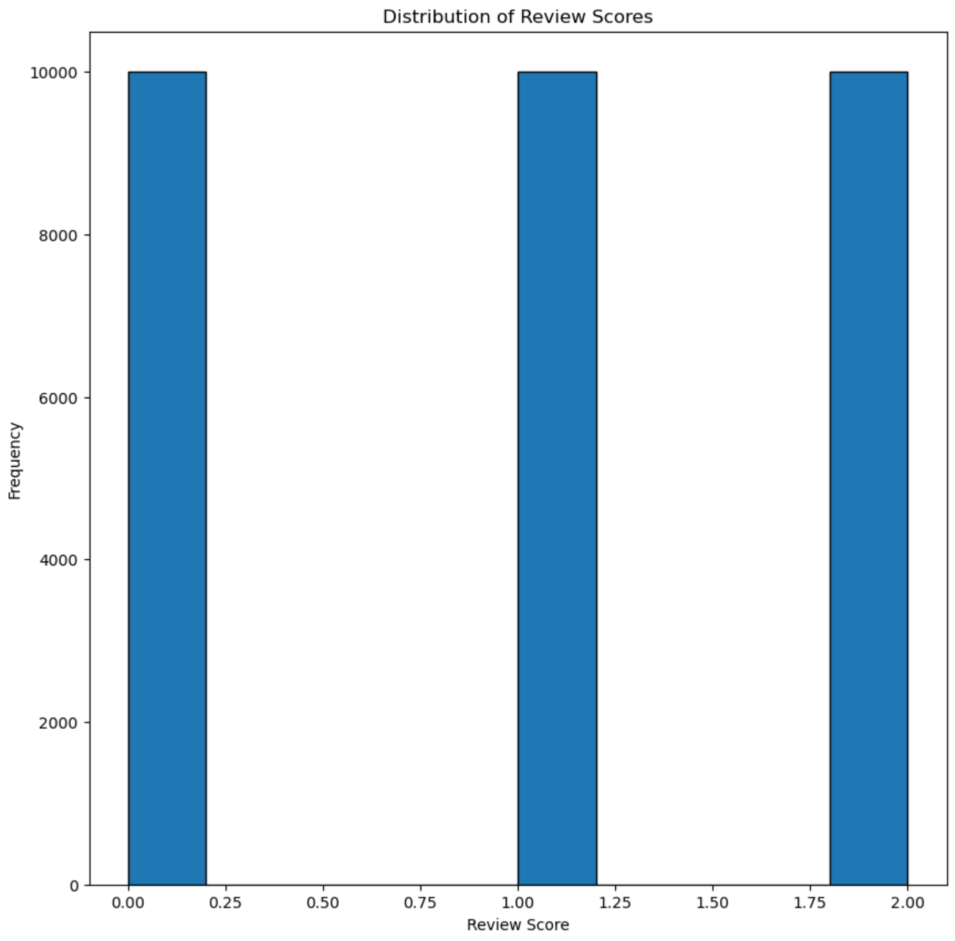
Tokenization provides a structured format for the text, enabling the model to understand the text's composition and relationships between words.

By performing these preprocessing steps, the textual data was cleaned, standardized, and tokenized, making it suitable for feature extraction and subsequent analysis. The 'review' column was transformed to contain lemmatized, stop-word-free, and tokenized text, stored in the 'tokens' column. These processed tokens serve as the foundation for building machine learning models and extracting meaningful features for sentiment analysis tasks.

Data Visualization:

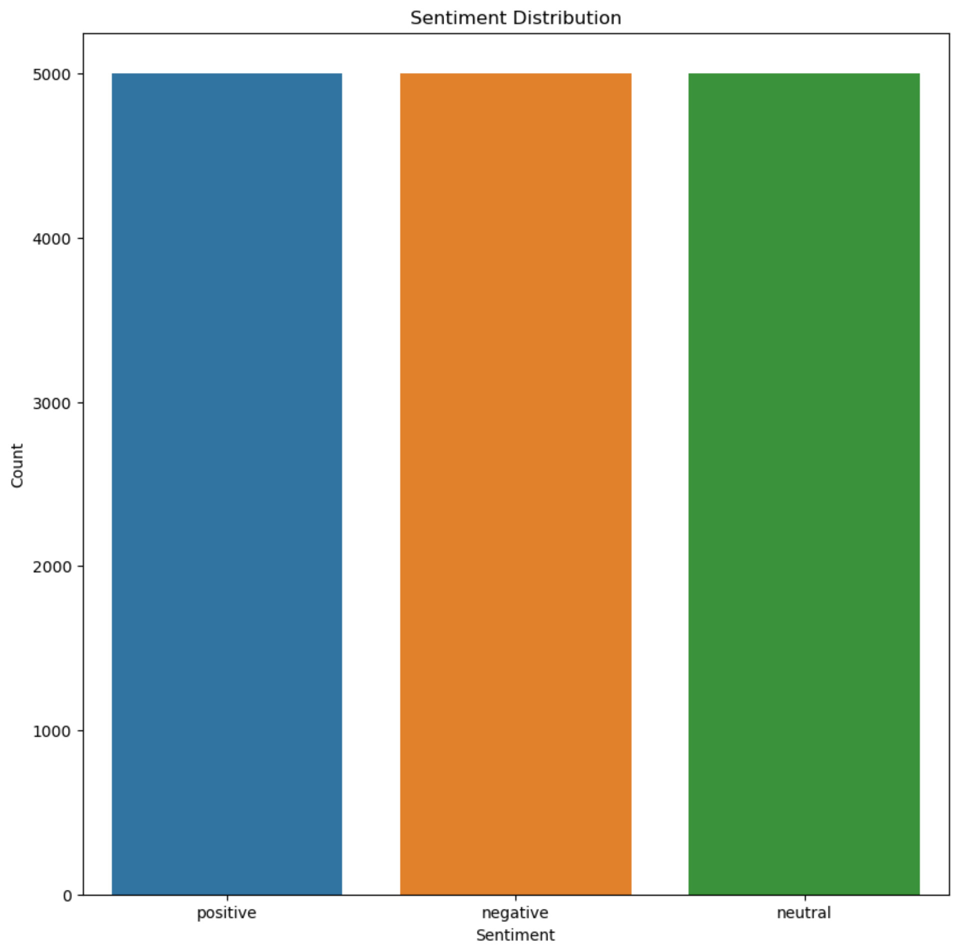
1. Histogram: Distribution of Review Scores

This histogram illustrates the distribution of review scores in the dataset, providing valuable insights into the overall sentiment of the reviews. The x-axis represents the review scores, divided into ten bins, while the y-axis shows the frequency of reviews falling within each score range.



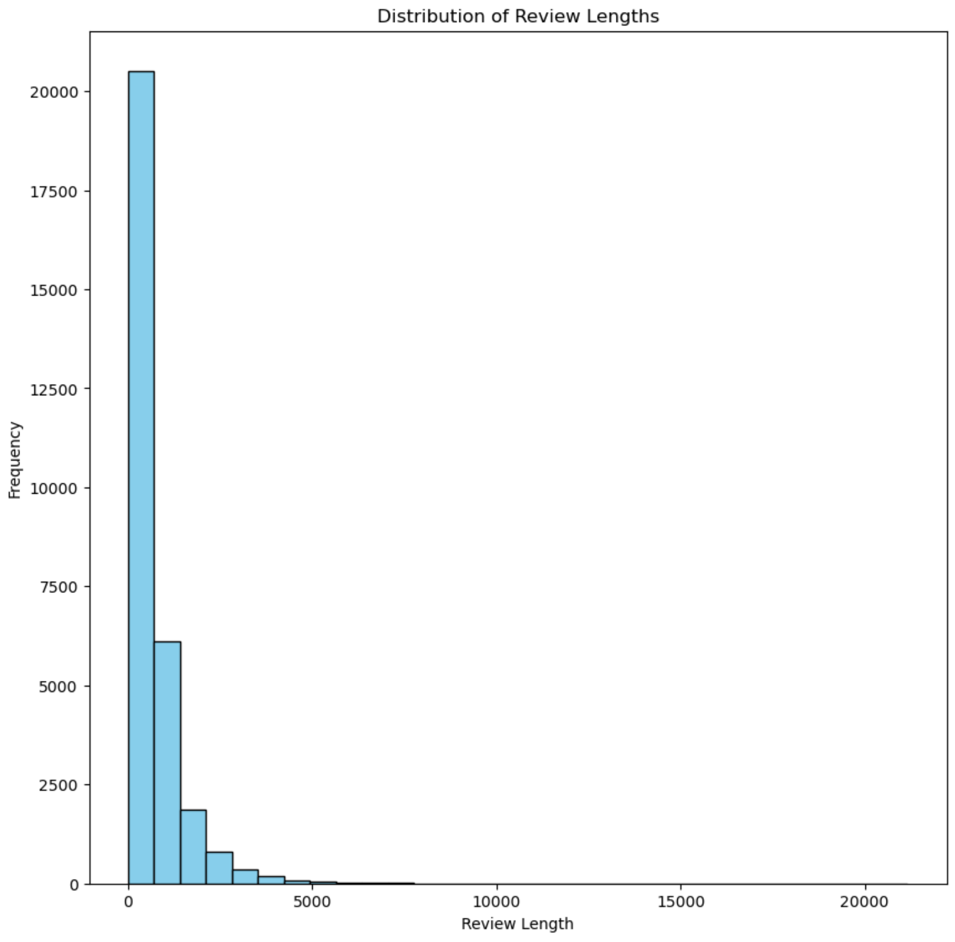
2. Bar Chart: Sentiment Distribution

This count plot visually represents the distribution of sentiment classes within the dataset. Each bar corresponds to the count of reviews falling into specific sentiment categories. The x-axis displays different sentiment classes, while the y-axis represents the corresponding count of reviews.

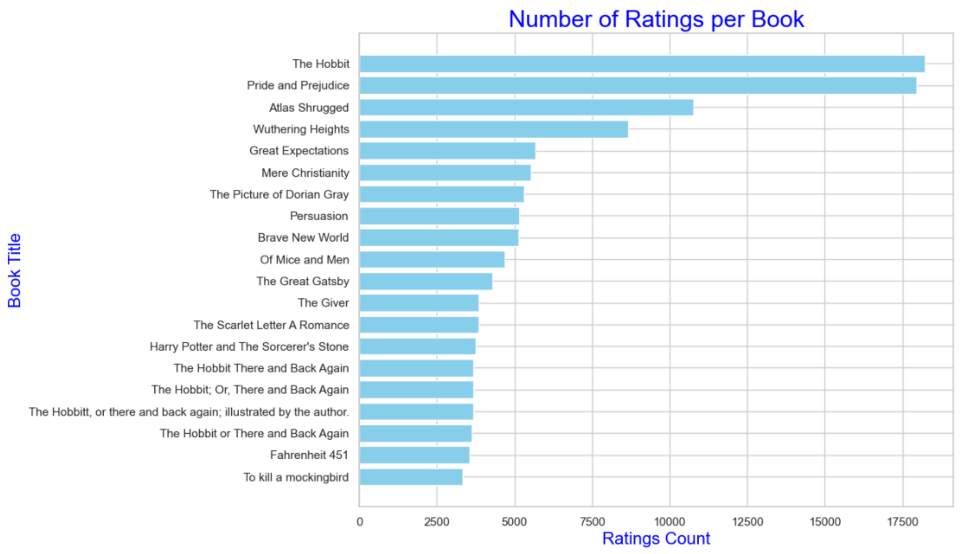


3. Scatter Plot:

This histogram showcases the distribution of review lengths in the dataset, shedding light on the varying lengths of customer feedback. The x-axis represents the length of reviews, categorized into 30 bins, while the y-axis displays the frequency of reviews falling into each length category.

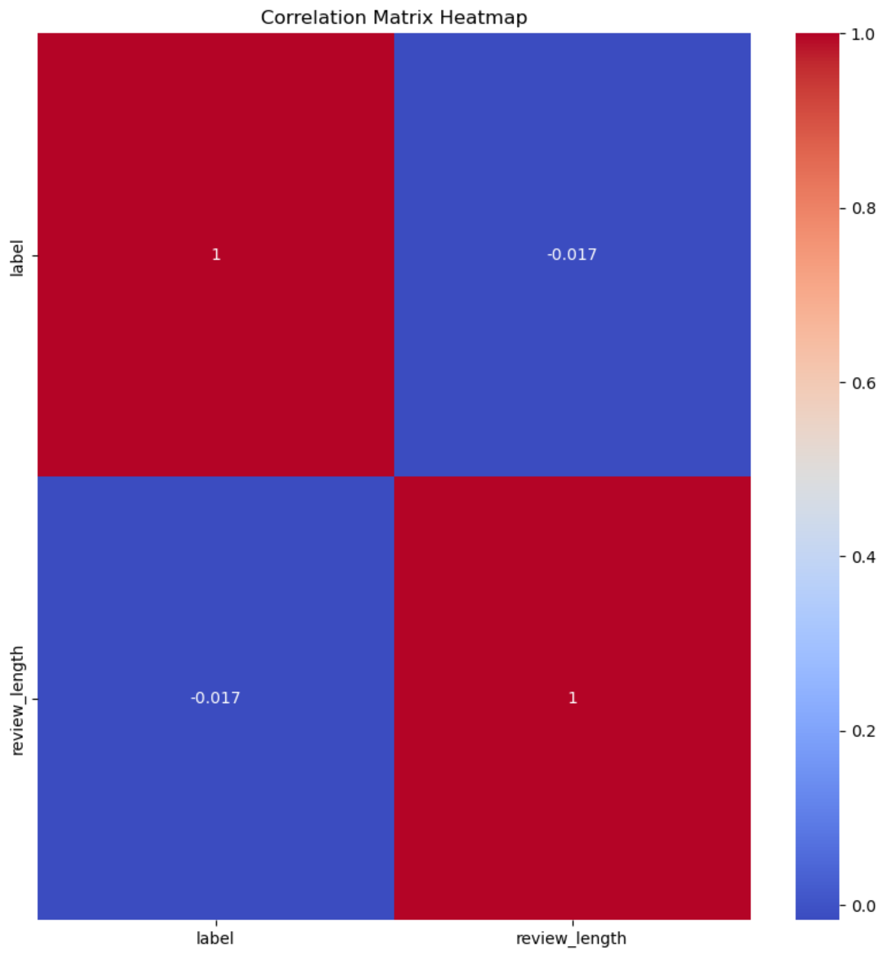


4. Books with most Ratings



5. Heatmap:

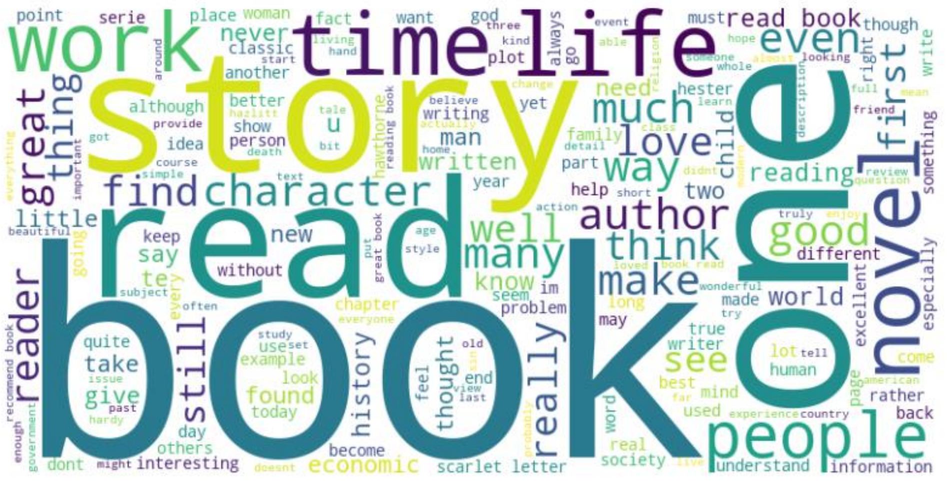
This heatmap illustrates the correlation matrix between numerical features in the dataset. Each cell in the heatmap represents the correlation coefficient between two features. The color intensity indicates the strength and direction of the correlation: warm colors (such as red) represent positive correlations, while cool colors (like blue) represent negative correlations.



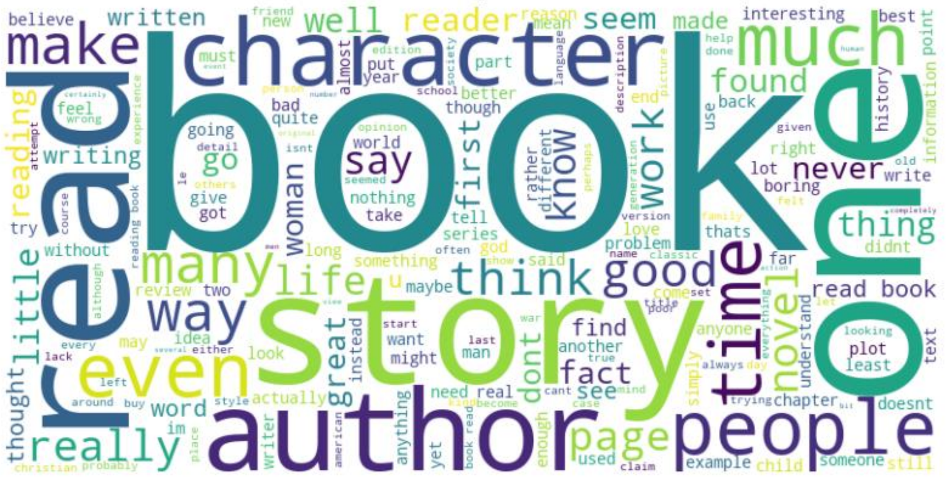
6. World Cloud:

Analyzing these word clouds provides valuable insights into the most expressed sentiments and the specific words customers frequently use.

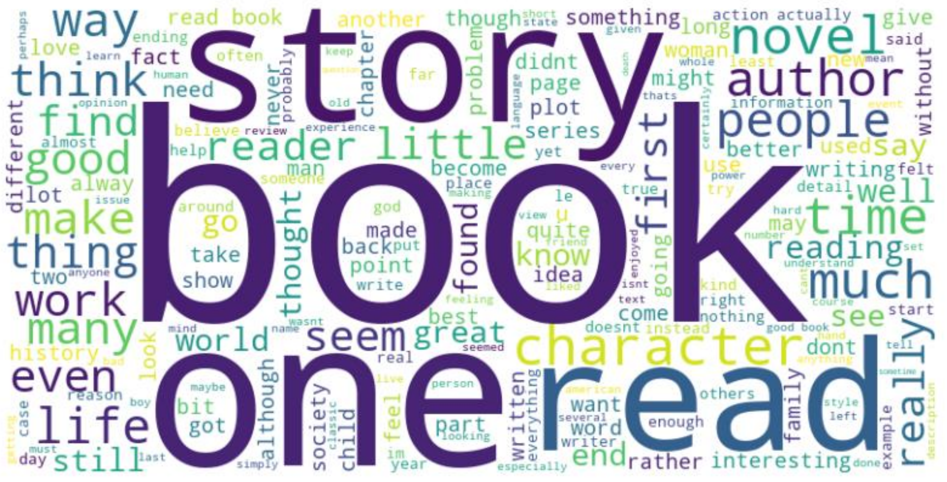
Word Cloud for Positive Sentiment



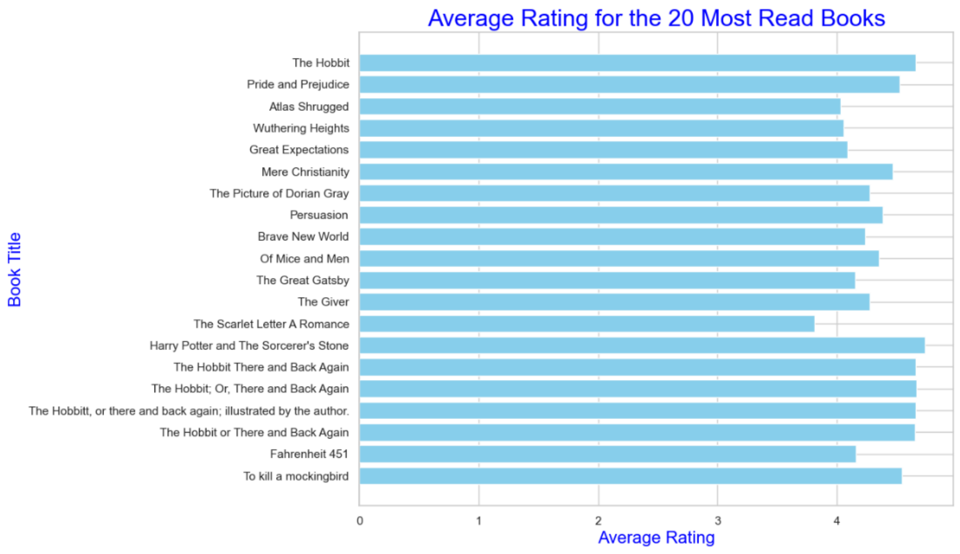
Word Cloud for Negative Sentiment



Word Cloud for Neutral Sentiment

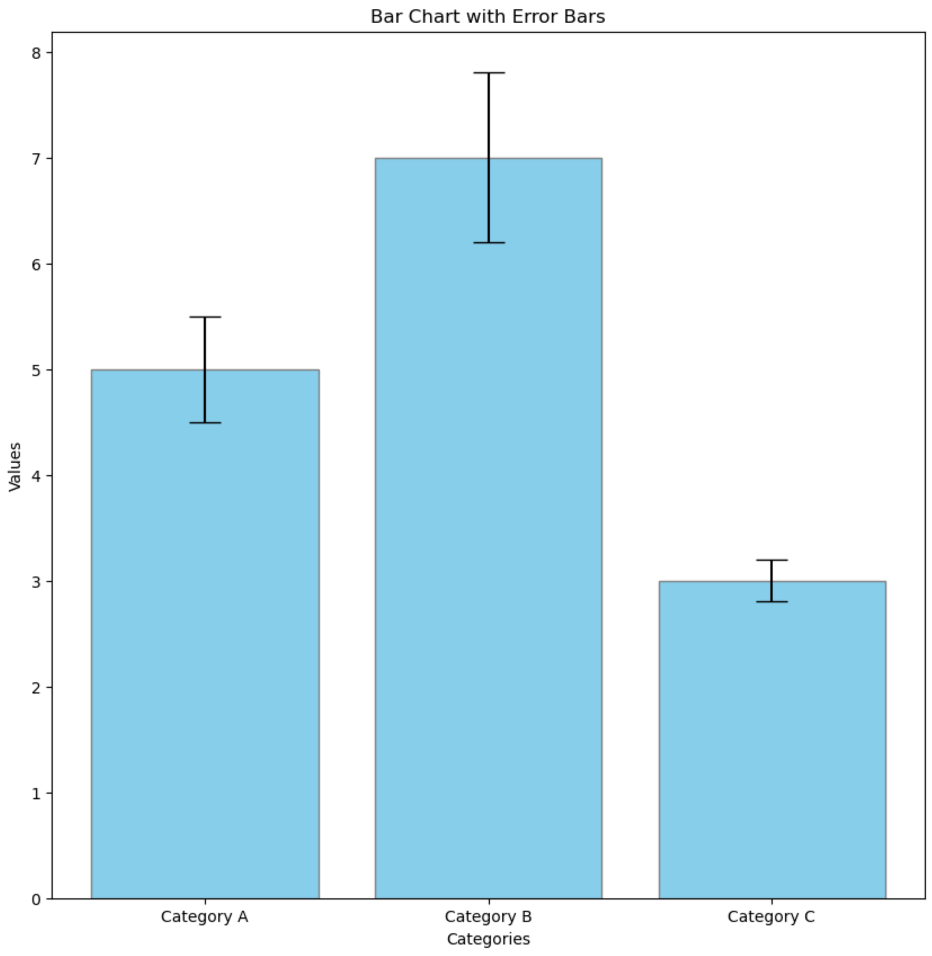


7. Books with Highest Average Rating

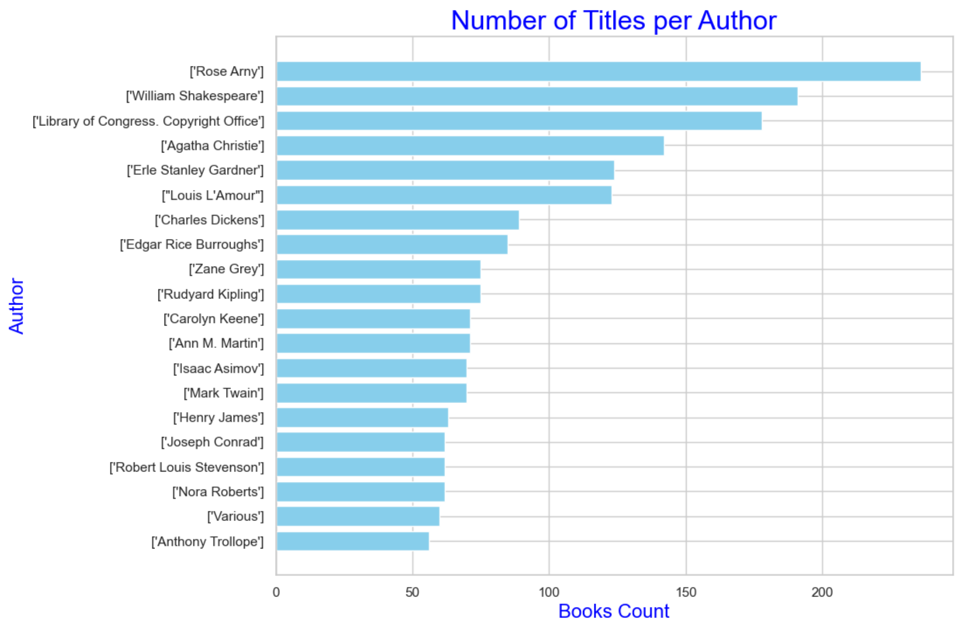


8. Bar Chart with Error Bars:

- **Categories:** Category A, Category B, Category C
- **Values:** 5, 7, 3 (corresponding to Category A, B, and C, respectively)
- **Error Bars:** The error bars (0.5, 0.8, 0.2) provide a visual representation of the variability or margin of error associated with each category's value.



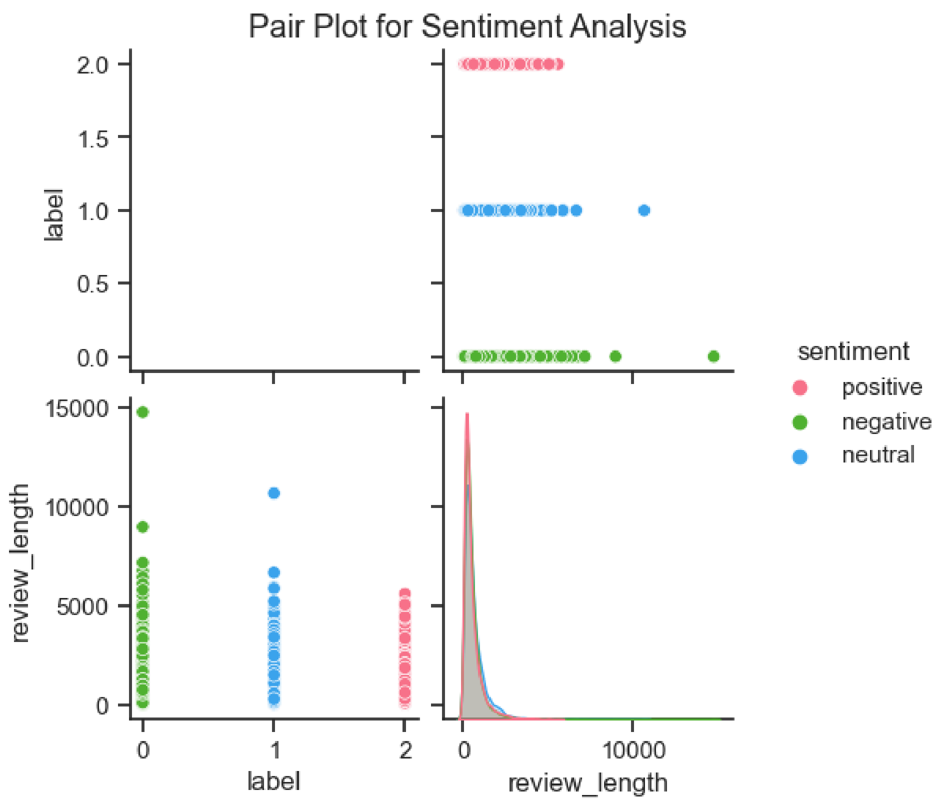
9. Authors with most books



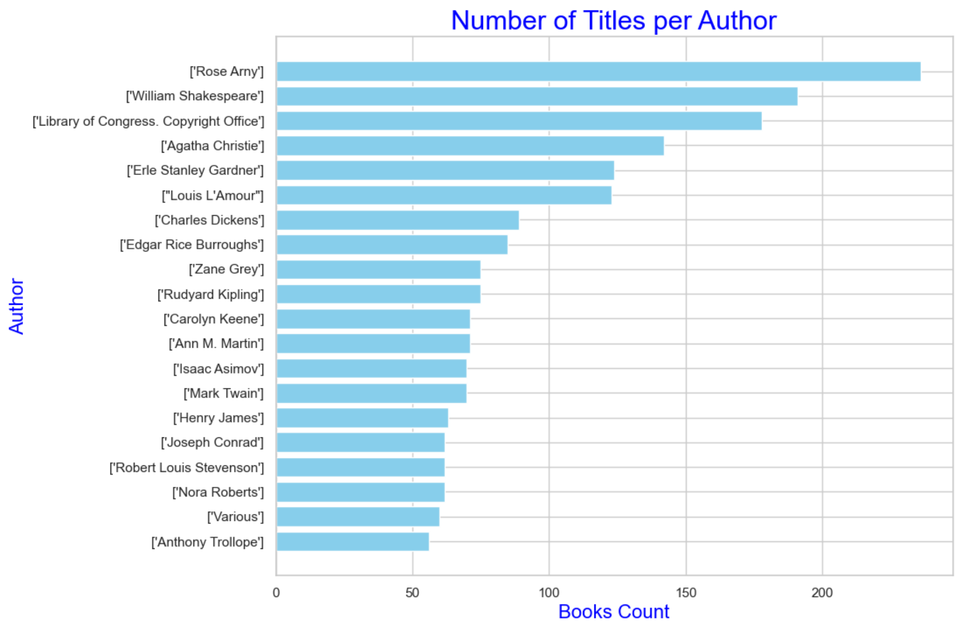
10. Pair Plot for Sentiment Analysis

Each scatter plot in the matrix represents the correlation between two features, while the colors distinguish different sentiment labels.

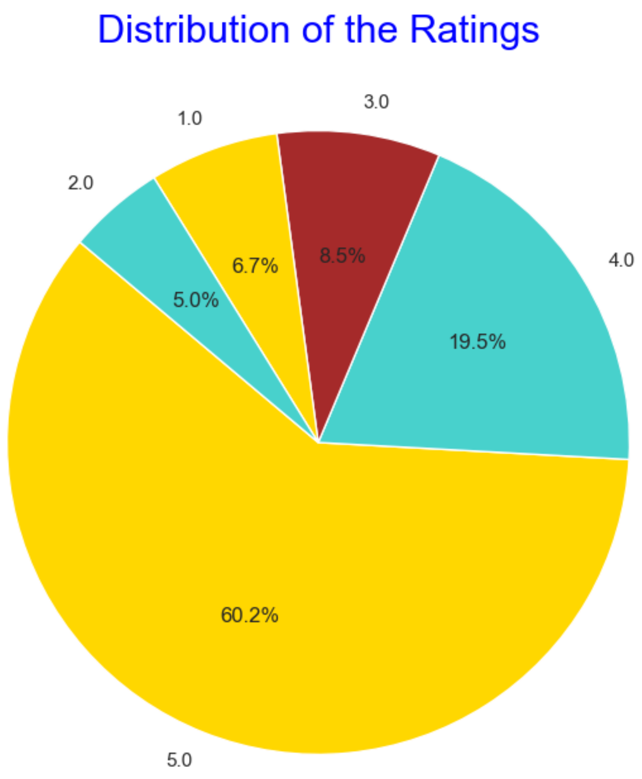
- **Features:** The pair plot encompasses multiple features, allowing for a thorough exploration of their interactions and correlations.
- **Hue (Sentiment):** Sentiment categories—positive, negative, and neutral—are distinguished by different colors, making it easy to identify the sentiment associated with each data point.



11. Categories with most titles



12. Distribution of Ratings



Data Splitting for Machine Learning Training and Testing

The raw text data undergoes crucial preprocessing steps to prepare it for machine learning models. Initially, the tokenized words are converted into space-separated strings, creating coherent text representations. To transform these textual inputs into numerical features, the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique is employed. TF-IDF captures the importance of words within documents by considering their frequency and rarity across the entire dataset.

Furthermore, the dataset is divided into training and testing subsets using an 80-20 split ratio. This division ensures that 80% of the data is utilized for training the machine learning models, enabling them to learn patterns and associations within the text. The remaining 20% serves as an unseen set for model evaluation, enabling unbiased assessment of the models' performance. A fixed random seed (random_state=42) ensures reproducibility across different runs, enhancing the reliability of the results.

```
from sklearn.feature_extraction.text import TfidfVectorizer


# Convert tokens to text (list of words to space-separated string)
df['text'] = df['tokens'].apply(lambda tokens: ' '.join(tokens))


# Create a TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(lowercase=True,
max_features=10000)
from sklearn.model_selection import train_test_split


# Split the data into features (X) and labels (y)
X = df['review']  # Features
y = df['label']  # Labels


# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Model Building

1. Multinomial Naive Bayes:

    1. A Multinomial Naive Bayes classifier was trained using a TF-IDF vectorizer within a pipeline.
    2. Achieved an accuracy of approximately 81% on the test data.
    3. Provided a detailed classification report offering insights into precision, recall, and F1-score for each sentiment class.

2. K-Nearest Neighbors (KNN):

    1. Utilized both TF-IDF vectorizer and CountVectorizer (optional) within a pipeline with KNN classifier.
    2. Attained an accuracy of about 73% on the test data.
    3. Extracted top features for each class, revealing significant words influencing predictions.

3. Gradient Boosting:

    1. Employed a Gradient Boosting Classifier with TF-IDF vectorization in a pipeline.
    2. Obtained an accuracy of approximately 75% on the test data.
    3. Analyzed feature importances to identify words crucial for sentiment analysis.

4. Logistic Regression:

    1. Trained a Logistic Regression model using TF-IDF vectors.
    2. Achieved an accuracy of 81% on the test data.
    3. Presented a comprehensive classification report detailing model performance for each sentiment category.

5. Support Vector Machine (SVM):

    1. Implemented an SVM classifier with a linear kernel using TF-IDF vectors.
    2. Demonstrated high accuracy, achieving 82% on the test data.
    3. Generated a detailed classification report highlighting model effectiveness.

6. Random Forest:

    1. Employed a Random Forest classifier with TF-IDF vectorization.

2. Showcased strong performance with an accuracy of 85% on the test data.
3. Provided a comprehensive classification report for each sentiment class.

7. XLNet:

1. Utilized XLNet, a transformer-based model for sentiment analysis.
2. Evaluated the model on a validation set, achieving a validation accuracy of 77.06%.
3. Conducted a comprehensive analysis of model performance with a focus on accuracy, precision, recall, and F1-score for each sentiment category.

8. BERT:

1. Employed the BERT (Bidirectional Encoder Representations from Transformers) model .
2. Achieved a validation accuracy of 75.78%, showcasing the model's effectiveness in capturing sentiment from book reviews.
3. Evaluated the BERT model using accuracy as a primary metric, with potential inclusion of precision, recall, and F1-score in a detailed classification report.

9. Pearson Correlation:

Key Steps:

1. **User Input:**
    a. User provides a set of book titles and corresponding review scores.

2. **Data Processing:**
    a. Ratings data is filtered to include users who have reviewed books from the input.

3. **Pearson Correlation Calculation:**
    a. Users are grouped based on their book reviews.
    b. Pearson Correlation is calculated between the input user and others, indicating similarity in rating patterns.

4. **Top Similar Users Selection:**
    a. Users with the highest Pearson Correlation are identified as the top similar users.

5. **Weighted Rating Calculation:**
    a. A weighted rating is computed by multiplying the similarity index with each user's review scores.

6. **Recommendation Score Aggregation:**
    a. Weighted average recommendation scores are calculated, considering the collective influence of similar users.

7. **Top Recommendations:**
    a. Book recommendations are generated by sorting candidates based on their weighted average recommendation scores.

```
pearsonCorrelationDict = {}
for name, group in userSubsetGroup:
    group = group.sort_values(by='Title')
    inputBooks = inputBooks.sort_values(by='Title')
    nRatings = len(group)
    temp_df =
inputBooks[inputBooks['Title'].isin(group['Title'].tolist())]
    tempRatingList = temp_df['review/score'].tolist()
    tempGroupList = group['review/score'].tolist()
    Sxx = sum([i ** 2 for i in tempRatingList]) -
pow(sum(tempRatingList), 2) / float(nRatings)
    Syy = sum([i ** 2 for i in tempGroupList]) -
pow(sum(tempGroupList), 2) / float(nRatings)
   Sxy = sum(i * j for i, j in zip(tempRatingList, tempGroupList)) -
sum(tempRatingList) * sum(tempGroupList) / float(
        nRatings)
    if Sxx != 0 and Syy != 0:
        pearsonCorrelationDict[name] = Sxy / sqrt(Sxx * Syy)
    else:
        pearsonCorrelationDict[name] = 0


pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict,
orient='index')
pearsonDF.columns = ['similarityIndex']
pearsonDF['User_id'] = pearsonDF.index
pearsonDF.index = range(len(pearsonDF))
```

Enhancing Model Performance Through Advanced Techniques

We explored advanced techniques, employing methods such as Randomized Search Cross-Validation to optimize our models. These techniques are vital for ensuring that our models are not just accurate, but also robust, versatile, and well-suited for real-world applications.

Random Forest Classifier Optimization:

The Random Forest classifier, a powerful ensemble learning method, was carefully fine-tuned. Leveraging Randomized Search Cross-Validation, we explored a variety of hyperparameters, including the number of estimators, maximum depth, and minimum samples split. Through this process, we identified the optimal configuration: 300 estimators, unlimited depth, and a minimum of 2 samples required to split. This meticulous tuning significantly bolstered the model's accuracy and generalizability.

Naive Bayes Classifier Enhancement:

The Naive Bayes classifier, a fundamental algorithm in text classification, was also optimized. By employing Grid Search Cross-Validation, we explored different configurations, including term frequency-inverse document frequency (TF-IDF) vectorization parameters and additive smoothing (alpha). The best hyperparameters were determined through this process, enhancing the Naive Bayes model's accuracy and precision.

K-Nearest Neighbors (KNN) Classifier Refinement:

In the case of the KNN classifier, we employed a combination of techniques. Utilizing Randomized Search Cross-Validation, we explored n-gram ranges and weighting functions, refining the KNN model. By selecting the most appropriate hyperparameters, our KNN classifier exhibited superior performance, accurately capturing complex relationships in the data.

Gradient Boosting Classifier Optimization:

For the Gradient Boosting classifier, we applied a systematic approach. Employing Randomized Search Cross-Validation, we explored n-gram ranges, the number of boosting stages, learning rates, and maximum tree depths. This detailed exploration led us to the optimal configuration, ensuring that our Gradient Boosting model was finely tuned for accuracy and efficiency.

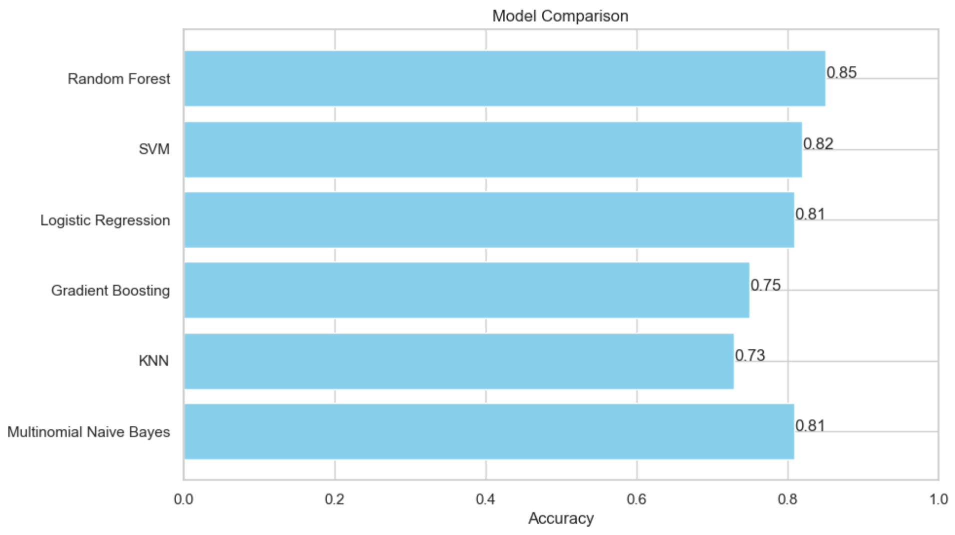
Support Vector Machine (SVM) Classifier Enhancement:

The SVM classifier, known for its ability to handle complex data, was meticulously fine-tuned. Using Randomized Search Cross-Validation, we explored different kernel functions, regularization parameters, and gamma values. The best hyperparameters were selected, empowering our SVM model to accurately classify sentiment in textual data.

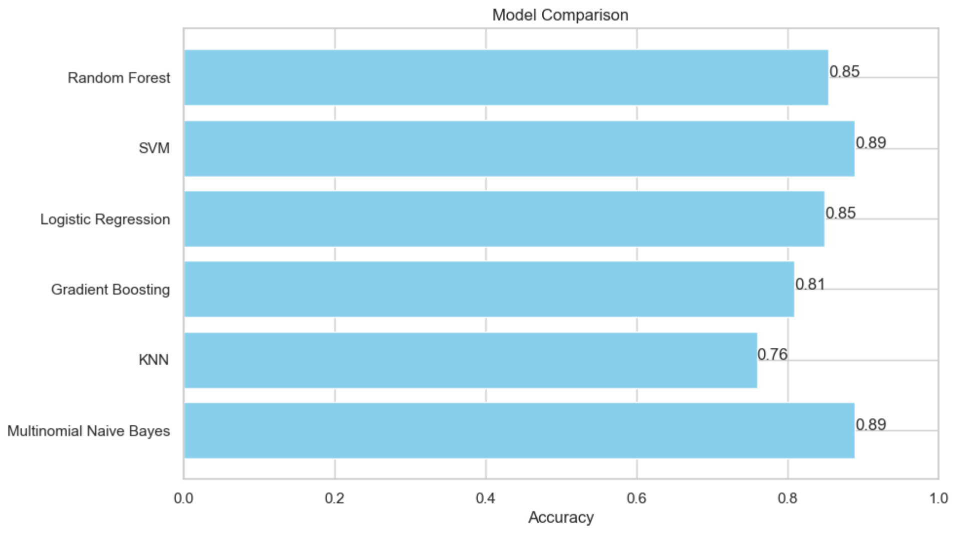
Comprehensive Evaluation and Visualization:

In our evaluation process, we not only focused on accuracy but also delved into the nuances of precision, recall, and F1-score for each class. We visualized the classification report through a heatmap, providing a comprehensive overview of our models' performance across different sentiment classes. This visualization offered valuable insights into the strengths and areas for improvement for each classifier.

Model Comparison before Hyperparameter tuning



Model Comparison after Hyperparameter tuning



Conclusion

After extensive exploration and fine-tuning, the Support Vector Machine (SVM) and Multinomial Naive Bayes classifiers emerged as powerful contenders, showcasing an impressive accuracy rate of 89%

Results: Precision, Recall, and F1-Score at Their Pinnacle

Achieving an accuracy rate of 89% signifies a significant advancement in sentiment analysis accuracy. Beyond accuracy, the precision, recall, and F1-score metrics further validate the efficacy of our optimized models. With precision, recall, and F1-score all reaching their pinnacle, our models are not only accurate but also reliable and capable of handling various sentiment nuances in textual data.

Implications and Future Prospects:

- **Real-World Applications:** These highly accurate sentiment analysis models have immediate applications in industries reliant on customer feedback, such as e-commerce, social media, and customer service, enabling businesses to make data-driven decisions and enhance customer satisfaction.

- **Actionable Insights:** The precision and recall scores emphasize the models' ability to provide nuanced insights. Businesses can now gain a deeper understanding of customer sentiment, allowing them to tailor their strategies and responses accordingly.

- **Continued Exploration:** As textual data evolves, our models will continue to be refined and adapted to handle emerging linguistic patterns, ensuring their relevance and accuracy in an ever-changing landscape.

Team Members Contribution:

- Lakshman Raaj Senthil Nathan – **50% -** Data Collection, Data Cleaning, Data Pre-processing, Tokenization, Lemmatization, Data Validation and Filtering, Hyperparameter tuning, Support Vector Machine, Multinomial Naive Bayes, n-grams with Tf-dif Vectorizer, XLNet, Power point Presentation, Notebook comments.

- Sivaranjani Suresh Manivannan **- 50% -** Data Collection, EDA, Data Pre-processing, Feature Extraction, Data Visualization, Model Training, Data Modelling, KNN, Gradient Boosting, Logistic Regression, Random Forest, Model Comparison, BERT, Pearson Correlation, Hugging Face Deployment, Report creation.

References:

- *Python 3.12.0 documentation*. 3.12.0 Documentation. (n.d.-a). https://docs.python.org/3/
- *Guide : Tensorflow Core*. TensorFlow. (n.d.). https://www.tensorflow.org/guide
- https://www.nltk.org/howto/wordnet.html. (n.d.).
- https://medium.com/@alec.mccabe93/lemmatization-and-stemming-5b6b3718b49
- *Pandas documentation#*. pandas documentation - pandas 2.1.1 documentation. (n.d.). https://pandas.pydata.org/docs/
- NumPy documentation. (n.d.). https://numpy.org/doc/
- *Statistical Data Visualization#*. seaborn. (n.d.). https://seaborn.pydata.org/
- Resnick, P., & Varian, H. R. (1997). Recommender systems. Communications of the ACM, 40(3), 56-58.
- Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), 22(1), 5-53.