Automated Review Rating System

Intoduction:

In the digital marketplace, customer reviews significantly influence purchasing decisions. Accurately predicting the rating of a product based on its textual review has become an important task in sentiment analysis and natural language processing (NLP). This project explores a deep learning-based approach to automatically predict numerical review ratings using text data. Specifically, it compares the performance of a a Bidirectional LSTM (BiLSTM) model with Bidirectional Gated Recurrent Unit to evaluate whether contextual learning in both directions improves prediction accuracy. The models are trained and tested on preprocessed customer review data to investigate their effectiveness in learning meaningful patterns and sentiments from raw text.

This system automates the process of rating reviews by:

- Cleaning and preprocessing raw review text.
- Visualizing review patterns and distributions.
- Creating a imbalanced dataset for fair model training.
- Splitting the data appropriately for training and evaluation.
- Applying Stopword removal and Lemmatization for cleaned text
- Applying vectorization and padding for better prediction

The resulting dataset and preprocessing steps lay the groundwork for building accurate and unbiased models capable of assigning star ratings to reviews thereby enhancing user experience and platform credibility.

Environment Setup:

To ensure smooth execution of the project and compatibility across systems, the following environment setup is recommended:

Python Installation

- Use **Python 3.10+**
- **Recommended:** Install Python via **Anaconda Distribution**
  Anaconda comes preloaded with most data science libraries and Jupyter Notebook support.

Required Python Libraries

The following libraries are essential for this project:

- pandas
- numpy
- matplotlib
- seaborn
- scikit-learn
- spacy
- tensorflow

Install them using:

```
pip install pandas numpy matplotlib seaborn scikit-learn spacy tensorflow
```

Additional Setup

- Download the English NLP model for spaCy:

```
python -m spacy download en_core_web_sm
```

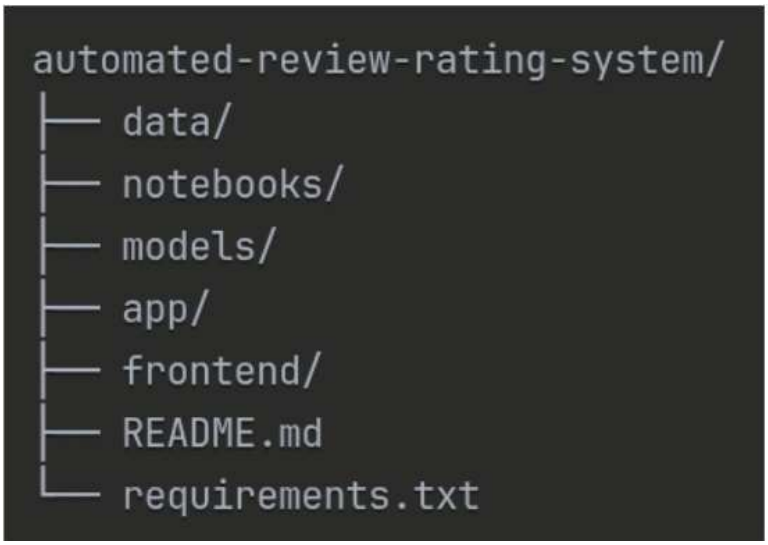
Github Project setup:

To manage version control and enable collaboration, a GitHub repository was created for the project.

Repository Name: https://github.com/hannafarsin/automatic-review-rating-system

Folder Structure:

The project is organized using a modular structure to ensure clarity, maintainability, and scalability:



Initial Commit:

- Added base folder structure.
- Created an initial `README.md` with a short description of the project.
- Included `requirements.txt` for setting up the Python environment.

1.Data Collection:

The dataset used for this project was sourced from **Kaggle**, specifically from the **Amazon Cell Phones and Accessories Reviews** dataset.

Dataset Overview:

- **Source**: Kaggle - Amazon Reviews
- **Dataset Link:** **https://github.com/hannafarsin/automatic-review-rating-system/blob/main/data/kaggle/reviews_output.csv**
- **Imbalanced Dataset Link:** **https://github.com/hannafarsin/automatic-review-rating-system/blob/main/data/kaggle/balanced_reviews_dataset3.csv**
- **Format**: CSV
- **Size**: 6,00,000+ reviews
- **Key Columns**:
  - `Review_text`: The main review written by the customer.
  - `Rating` (or `overall`): The star rating given by the user (ranging from 1 to 5).

Purpose of Data Collection:

The raw data serves as the foundation for training an automated system that can predict a review's rating purely from its text. Since the original dataset is

imbalanced (i.e., more 5-star reviews than 1-star), additional balancing steps were planned as part of the preprocessing.

Dataset Preview:

[2]:

	Text	Score
0	I have bought several of the Vitality canned d...	5
1	Product arrived labeled as Jumbo Salted Peanut...	1
2	This is a confection that has been around a fe...	4
3	If you are looking for the secret ingredient i...	2
4	Great taffy at a great price. There was a wid...	5
...	...	...
607443	cheap product its not a microwave safe	2
607444	it broked when the food is in on itnot one pla...	2
607445	full plate size is too small	2
607446	plates size is small bowl is too smallquality ...	2
607447	ok	2

607448 rows × 2 columns

2. Data Preprocessing:

2.1 Initial Data Inspection

Before applying text preprocessing techniques, the dataset was examined for structural issues:

Missing (null) values:

Columns like `reviewText`, `Rating` were checked.

df.isnull().any()

```
review_text     True
Rating         False
```

df.isnull().sum()

```
review_text    9
Rating         0
```

`Review_text` were removed.

```
df_no_mv=df.dropna(subset=[''reviewText'], inplace=True)
df_no_mv.isnull().sum()
```

```
review_text        0
Rating             0
```

Duplicate entries:

Exact duplicates were identified and dropped to avoid model bias.

```
df_no_mv.duplicated().sum()
```

```
0
```

Remove Conflicting Reviews(Same text,Different Ratings):

```
# Check how many reviews have conflicting ratings
conflict_counts = (df_no_mv.groupby('review_text')['Rating'].nunique()
    .reset_index().rename(columns={'Rating': 'unique_ratings'}))
# Get texts that have more than 1 unique rating
conflicting_reviews = conflict_counts[conflict_counts['unique_ratings'] > 1]['review_text']
# Filter original df for only conflicting reviews
conflict_df = df[df['review_text'].isin(conflicting_reviews)]
# Group by review_text to see all associated ratings
conflict_summary = conflict_df.groupby('review_text')['Rating'].unique().reset_index()
#  Show a few rows
conflict_summary.head(10)
```

[18]:

	review_text	Rating
0	#NAME?	[3, 2]
1	<a href="http://www.amazon.com/gp/product/B004...	[3, 4]
2	A Ming Dynasty (AD 1368 - 1644) creation belie...	[5, 4]
3	A few years ago, my cat was diagnosed with foo...	[5, 4]
4	After trying both the Chipotle as well as the ...	[5, 4]
5	Any of us who grew up during the U.S. space ag...	[4, 5]
6	Before the details: The Planters big nut bars...	[5, 4]
7	Bought this coffee in Europe when I was travel...	[4, 5]
8	Buffalo Bills Premium Snacks I real...	[3, 5]
9	Coffee drinks aren't carbonated, so it's odd t...	[2, 1]

2.2 Text Cleaning and Normalization:

In this step, the raw review text was cleaned and standardized to reduce noise and prepare it for further processing.

 # 1. Pre-split cleaning: punctuation, emojis, spaces

```
def basic_clean(text):
    text = str(text).lower()
    text = re.sub(r'[' + string.punctuation + ']', '', text)        # remove punctuation
    text = re.sub(r'[^\x00-\x7F]+', '', text)                       # remove emojis / non-ASCII
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)  # remove URLs
    text = re.sub(r'<.*?>', '', text)                               # remove HTML tags
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)                      # remove special characters
    text = re.sub(r'\s+', ' ', text).strip()                        # remove extra spaces, tabs, etc.
    return text
df_cleaned['review_text'] = df_cleaned['review_text'].apply(basic_clean)
```

Cleaning Steps Applied:

1. Lowercasing

```
text = str(text).lower()
```

- All text was converted to lowercase to maintain consistency.
- Example:
  `"This PHONE is Great!"` → `"this phone is great"`

2. Removing URLs and HTML Tags

```
text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)  # remove URLs
text = re.sub(r'<.*?>', '', text)                               # remove HTML tags
```

- Any embedded links or HTML elements were removed using regular expressions.
- Example:
  `"Check it out: <a href='...'>Link</a> "` → `"check it out"`

3. Removing Emojis and Special Characters

```
text = re.sub(r'[^\x00-\x7F]+', '', text)                       # remove emojis / non-ASCII
text = re.sub(r'[^a-zA-Z0-9\s]', '', text)                      # remove special characters
```

- Emojis, symbols, and non-ASCII characters were stripped from the text.
- Example:
  `"Great phone 😍✨!"` → `"great phone"`

4. Removing Punctuation and Numbers

```
text = re.sub(r'[' + string.punctuation + ']', '', text)          # remove punctuation
```

- Punctuation marks (e.g., `! ? , .`) and digits were removed to simplify the text.
- Example:
  `"Rated 10/10!!!"` → `"rated"`

5.Check Duplicate values after text cleaning :

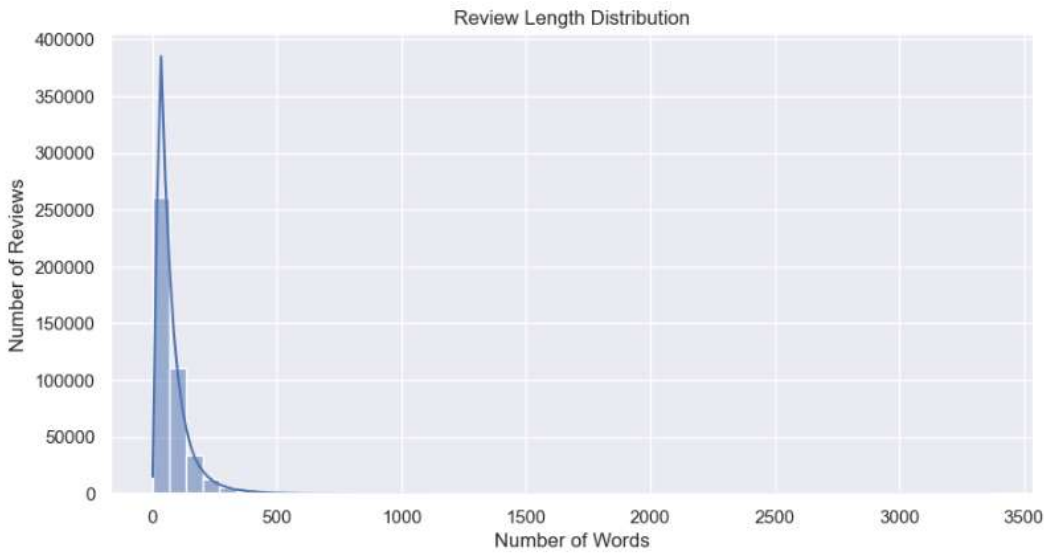
```
df_filtered.duplicated().sum()
```

```
503
```

Remove duplicate values

```
#drop duplicated now
df_filtered = df_filtered.drop_duplicates(subset='review_text', keep='first').reset_index(drop=True)
```

6. **Review Length Filtering**



- Very short reviews (fewer than 3 words) and excessively long ones (e.g., over 500 words) were removed.
- This helped eliminate outliers and non-informative entries.

Justification for Review Length Filtering

The histogram above illustrates the **distribution of review lengths** based on word count. It shows a **heavily right-skewed distribution**, where:

- The **majority of reviews contain fewer than 300 words**, with a sharp peak under 50 words.
- A **long tail of outliers** includes reviews with extremely high word counts, some exceeding **3500 words**.

To improve data quality and processing efficiency:

- **Very short reviews** (less than 3 words) were removed, as they offer limited semantic information.
- **Excessively long reviews** (e.g., over 500 words) were filtered out to eliminate outliers that may introduce noise or slow down training.

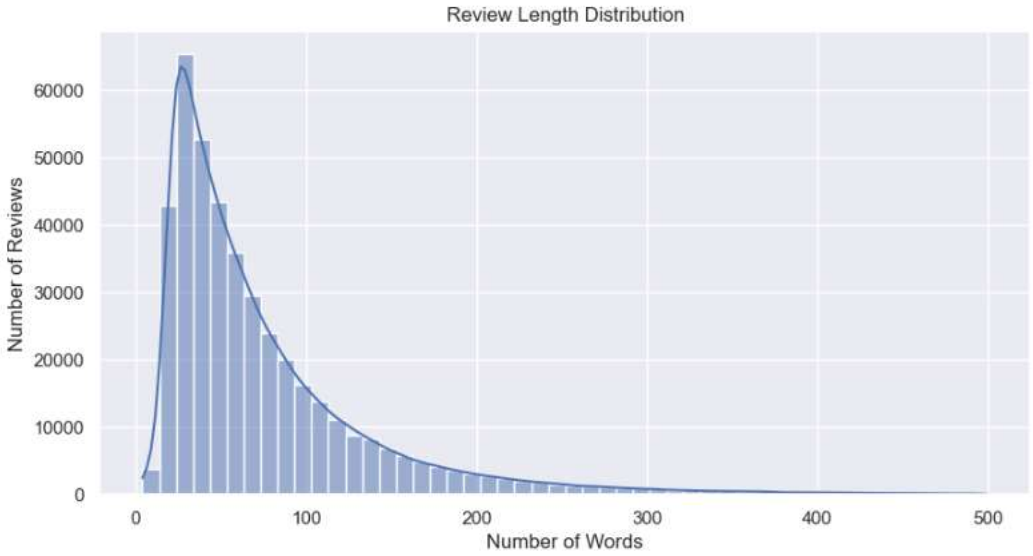
This step ensures a more **uniform distribution**, making the dataset more suitable for vectorization and model training.

After Filtering:

- Very short reviews (fewer than 3 words) and excessively long ones (e.g., over 350 words) were removed.
- This helped eliminate outliers and non-informative entries

```
df_filtered = df_cleaned[
    (df_cleaned['review_length'] > 3) &
    (df_cleaned['review_length'] <= 500)
]
print("Before filtering:", len(df_cleaned))
print("After filtering:", len(df_filtered))
```

```
Before filtering: 430456
After filtering: 427624
```



After applying the text cleaning and length-based filtering:

- Reviews with **fewer than 3 words** and those with **excessive lengths (over 500 words)** were removed.
- The remaining dataset shows a **more uniform and meaningful distribution** of review lengths, as visualized in the updated histogram above.
- Most reviews now fall between **10 to 150 words**, providing enough context while avoiding noise from overly short or verbose entries.

3. Creating a Imbalanced Dataset

Why Imbalance the Dataset?

In real-world applications such as online review systems, customer feedback often follows an **imbalanced rating distribution**. For instance, most users tend to leave 4-star or 5-star reviews when they are satisfied, while only a small fraction leave 1-star or 2-star reviews when dissatisfied.

Building and evaluating models **only on balanced datasets** can lead to overly optimistic or misleading performance metrics. In contrast, using an **imbalanced dataset during evaluation** helps to:

- • **Simulate real-world conditions**
- • Reveal how the model handles rare classes (like 1-star ratings)
- • Uncover biases toward majority classes (e.g., 5-star reviews)
- • Validate the robustness of models trained on balanced data

The Challenge
In imbalanced datasets:

- • Classifiers tend to **favor the majority class**
- • Metrics like **accuracy** become less reliable (e.g., always predicting "5-star" gives high accuracy)
- • **Minority classes are underrepresented**, yet are often the most important (e.g., critical feedback)

Rating Distribution across 1 through 5 stars are:

```
df_filtered['Rating'].value_counts()
```

```
Rating
5    249661
4     55649
3     50567
1     36045
2     35170
```

As shown, over **60% of the reviews** are either 4 or 5 stars, while the 1- and 2-star reviews together make up **less than 15%**. This imbalance is typical in real-world data but can lead to models that underperform on minority classes (e.g., 1- and 2-star reviews).To simulate and

analyze the impact of a controlled imbalance, we constructed a **custom imbalanced dataset** consisting of **1,00,000 reviews**, following the distribution:.

Imalancing Strategy:

To simulate real-world bias and evaluate model robustness under class imbalance, we constructed a **custom imbalanced dataset with exactly 10,000 reviews**, using the following rating distribution:

Rating	% of Total	Sample Count
☆ 1	10%	1,0000
☆ 2	15%	1,5000
☆ 3	25%	2,5000
☆ 4	30%	3,0000
☆ 5	20%	2,0000
Total	**100%**	**100,000**

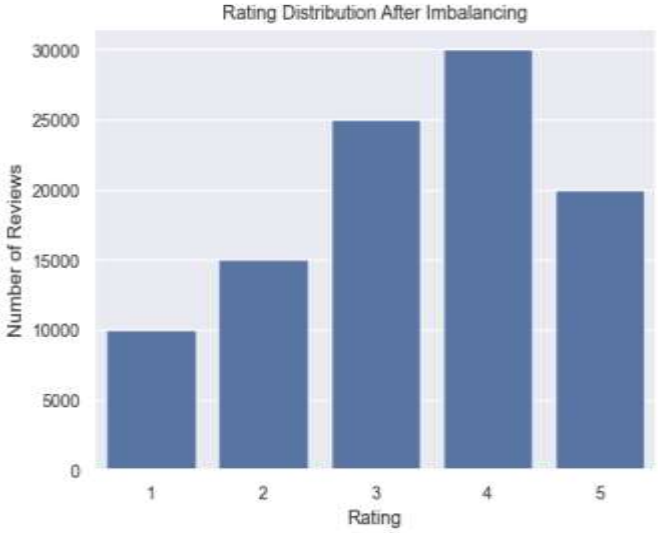
Code:

```
#imbalanced dataset distribution strategy
#  Define target class counts for imbalance
target_counts = {
    1: 10000,
    2: 15000,
    3: 25000,
    4: 30000,
    5: 20000
}
# Sample from remaining dataset based on distribution
df_imbalanced = pd.DataFrame()

for rating, count in target_counts.items():
    subset = df_remaining[df_remaining['Rating'] == rating]
    sampled = subset.sample(n=count, random_state=42)
    df_imbalanced = pd.concat([df_imbalanced, sampled], axis=0)

#  Shuffle final imbalanced dataset
df_imbalanced = df_imbalanced.sample(frac=1, random_state=42).reset_index(drop=True)

# Step 7: Save to CSV
df_imbalanced.to_csv("deep_imbalanced_dataset.csv", index=False)

# Optional: Check counts
print("Imbalanced class distribution:\n", df_imbalanced['Rating'].value_counts())
```

Important Note: To ensure independence from the model's training phase, the samples used to create this imbalanced dataset were taken only from the portion of the dataset not used in the balanced training set. This avoids data leakage and ensures a clean, fair evaluation of the model's performance on previously unseen examples**.**

```
used_balanced = df_balanced.index

df_remaining = df_filtered.drop(index=used_balanced, errors='ignore')
```

Imbalanced Rating Distribution



The bar chart above confirms that the dataset has been successfully balanced. Each rating class (from 1 to 5 stars) now contains exactly **2,0000 reviews**, ensuring equal representation across all categories.

This balanced distribution addresses the skew present in the original dataset and helps:

- Prevent model bias toward majority classes
- Improve accuracy for underrepresented ratings
- Ensure fair and consistent performance during training and evaluation

This clean and uniform dataset is now ready for final preprocessing and modeling steps.

Display the statistics of the word count after balancing:

```
df['word_count'] = df['review_text'].apply(lambda x: len(str(x).split()))
#display the statistics of the word count
min_word_counts = df.groupby('Rating')['word_count'].describe()
print(min_word_counts)
```

```
count       mean             std       min   25%   50%    75%    max
Rating
1       10000.0  76.703500  61.420067  5.0  35.0  59.0   97.0  499.0
2       15000.0  77.265000  74.149615  4.0  29.0  54.0   98.0  500.0
3       25000.0  87.303160  79.162376  4.0  34.0  59.0  111.0  500.0
4       30000.0  85.589767  70.755941  5.0  37.0  63.0  110.0  497.0
5       20000.0  70.864350  59.673597  4.0  32.0  52.0   88.0  498.0
```

Display Sample Review per Rating

```
#display 3 full sample reviews per rating
for rating in sorted(df['Rating'].unique()):
    print(f"\n ☆ Rating {rating}")
    reviews = df[df['Rating'] == rating].tail(3)
    for _, row in reviews.iterrows():
        print(f"- {row['review_text']}")
```

```
☆ Rating 1

- this instant coffee brand is way overrated and doesnt even compare to the taste and aroma of tasters choice instant coffee french roast


- were terribly disappointed this was the second time we ordered this product the first 5lb bag was incredible we snarfed them all down the olives were firm and ful l of flavor had a nice greek olive scent the second one was terrible they tasted r ancid and smelled terrible we ended up throwing them all away we were afraid wed g et sick we will not be ordering again very disappointed that the product was not e ven edible


- the last shipping i got the tea is so horrible it is like the 3rd time that you put water init is so weak it hardly tasted like green tea that i used to get i don t know what wrong with it i ordered 6 boxeseach so it took me a few months to fini sh all 6 boxes with 2nd package with 6 boxes those a received about a couple month s ago they are so terrible it almost does not smell like a real tea any more


☆ Rating 2

- when i received my bag nearly all of the flowers were broken and fractured leavi ng a lot of dust and very little whole flowers lots of stems too


- i have purchased this product from my vet howeverthe vet is a long drive to purc hase this item the first time i purchased this product from amazon it looked smell ed and felt like the product i had purchased from my vet however this time they sm ell different they look and feel different from the product i purchased the first time either they have changed something in the process or these have expired my do g still likes them she has not gotten sick so they must be ok


- my husband is a type2 diabetic who enjoys sweet treats so ive tried to look for sugarfree products that will agree with him and satisfy his sweet tooth he loves c hocolate brownies so after reading most of the positive reviews here i decided to buy the sixpack brownie mix and try it outmy husband liked the taste though he adm itted it was nowhere near as rich and flavorful as the ghirardelli brownie mixes w hich i use occasionally at home taste aside he also experienced bloating and exces sive gas initially i thought this was due to something else he ate but when he ate another brownie similar effects convinced us it was due to the maltitol ingredient in the mixi think this mix will be ok for those who are not averse to maltitol but i would not advise it for those who are averse to it i guess my search for the per fect sugarfree brownie mix continues
```

☆ Rating 3

- ive tried three different flavors of starbucks natural fusions coffee so far eve n though i love cinnamon and sometimes add some ground cinnamon to my coffee groun ds to flavor regular coffee i didnt much care for the flavor of starbucks cinnamon flavored coffee this vanilla flavor is pretty good however though i will say that i dont like it enough to buy it again the real winner of the three is the caramel flavor while drinking it wont make you think youre eating a caramel candy it does have a very pleasing flavor whether consumed hot or cold thats the only flavor of the three that i have purchased repeatedly and will purchase againas for the under lying coffee flavor of this particular flavor i think it suffices to say that it t astes like a medium roasted starbucks coffee i happen to like starbucks coffee bea ns but i know some people dont care for the brand if youre a fan of starbucks this roastflavor is consistent with what youve come to expect from them

- my daughter begged for this case but it wasnt what we expected when we received it it is very cheaply made and it feels cheap

- i bought this product to hold my galaxy s3 while running i saw other reviewers s ay that it would work with the phone in a uag case which is what i have it does fi t but is very tight sometimes while running it might hit the volume keys up and do wn which can get annoying but i can cope with that one other issue i have is that the sweat gets all over my phone since there is just a thin layer of neoprene in b etween my arm and the phone a piece of plastic between the phone and the fabric co uld probably solve that issue but the worst part of this product is that it is itc hy there isnt enough padding in place near the velcro areas and when the velcro co mes over or across it becomes very itchy because of these issues i would probably not recommend it maybe try to find another product that has better padding

☆ Rating 4

- the source of protein in hemp seeds is much easier to ingest than whey the flavo r is very good rather nut like and is very good on whole grain cereals ive cut dow n on red meat and use the hemp seed product to keep a handle on the protein that m y body is asking for im 61 years old and am an active outdoors person i do not tak e junky supplements of any kind in tablet or capsule form this source of protein i s natural and delicious

- firm sweet and tasty if you cook it right firm means not marshy or soggy a quali ty youd like to have in rice easy to cook it doesnt take long above all what i lik e most about botan is the pure quality in the taste almost crystallike it mustve b een grown in good quality water

- of course these dont taste as great as the cereal bars but they are way better t han most protein bars this is the only flavor ive tried and theyre not fabulous bu t definitely edible im looking forward to trying the chocolate peanut butter flavo r and hey ive lost 7 pounds

☆ Rating 5

- i was glad to see a variety of flavors as well as brands in the assortment the o nly thing i didnt like was the packaging all 35 pieces thrown in a plain white box

```
i had to take them out of the box and sort them to see what was there overall a ve
ry good product and i would buy it again


- this is by far the best canned tuna i have ever eaten i like the smaller cans be
cause they are a perfect single serving size ive had more expensive canned tuna bu
t none betterican eat it right out of the can after draning no salt no nothinits t
hat good and it is consistent from can to can


- our dog savannah loves this food first time i bought it she ate everything in th
e bowl i love that its organic and doesnt have wheat products id recommend it to a
nyone who cares about their dogs health
```

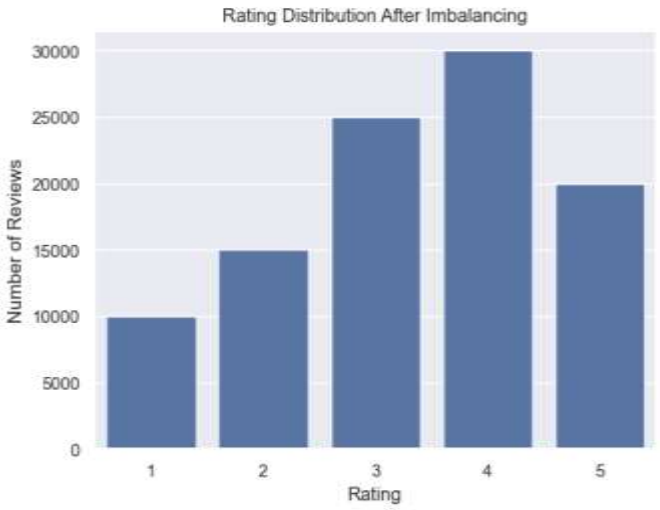
4. Data Visualization

To better understand the characteristics of the reviews and guide preprocessing decisions, several visualizations were created. These plots offer insights into rating distribution, review length, and review content.

4.1 Review Rating Distribution (Before Imbalancing)

Insight: The original dataset was highly imbalanced, with over 2,50,000 reviews rated 5 stars, while 1- and 2-star reviews were much less frequent. This confirmed the need for dataset balancing before training.

4.2 Review Rating Distribution (After imbalancing)

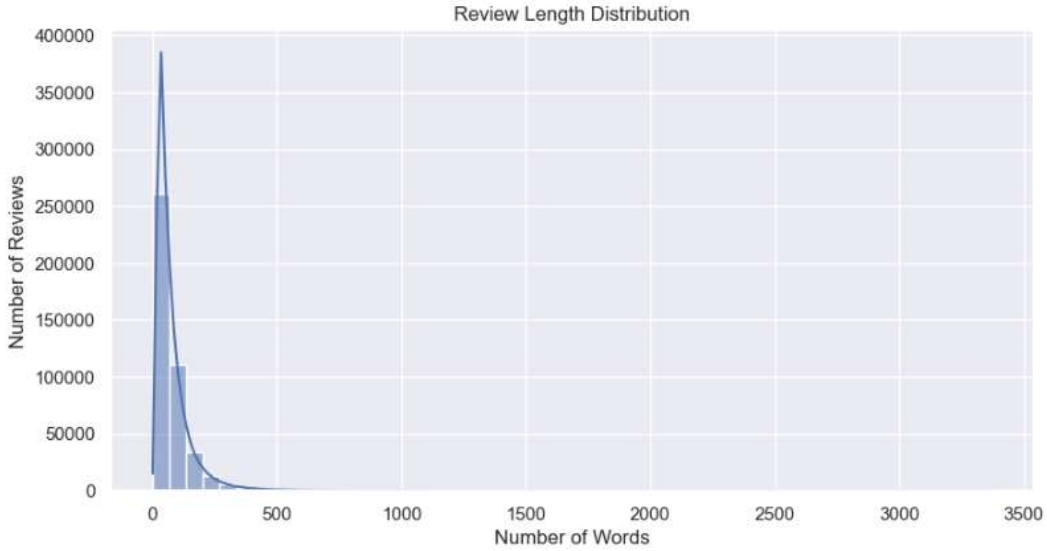


The bar chart clearly illustrates the **custom imbalanced distribution** of product review ratings:

- **4-star reviews** are the most common, making up **30%** of the dataset (3,000 samples).
- **3-star reviews** contribute **25%** (2,500 samples), followed closely by **5-star reviews** at **20%** (2,000 samples).
- **Lower ratings** (1-star and 2-star) make up a smaller portion of the dataset—**10% and 15%**, respectively.

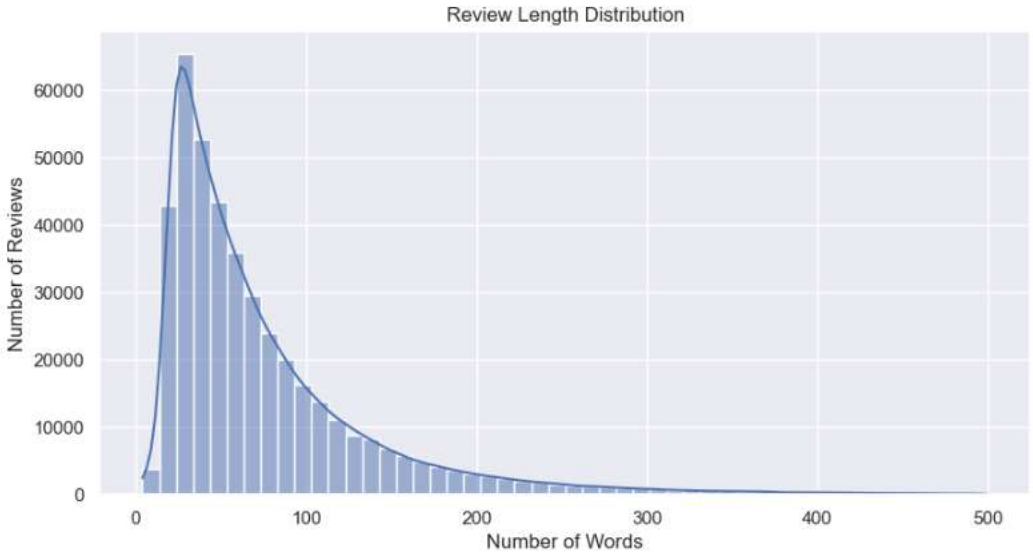
This imbalance reflects a **real-world scenario** where users are more likely to leave moderate-to-high ratings than low ratings

4.3 Review Length Distribution (Before Filtering)



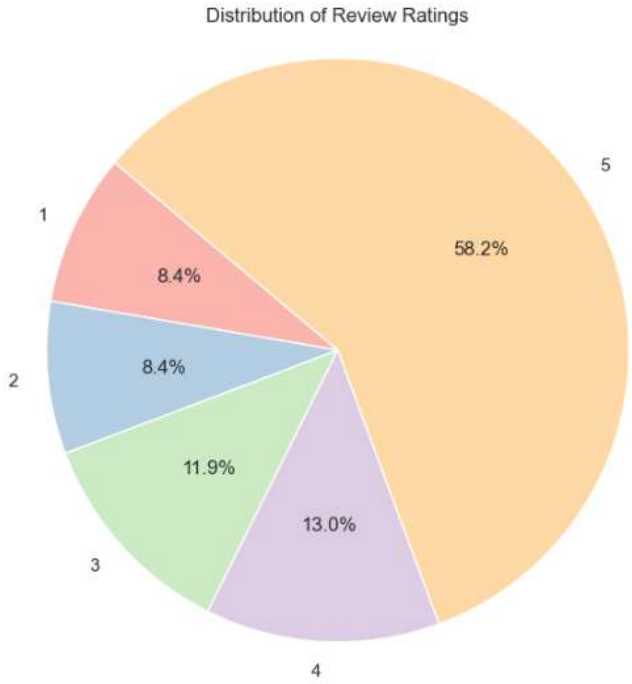
Insight: Most reviews are under 100 words, but some extend beyond 3,500 words. The long tail justified filtering out extremely long reviews as outliers.

4.3 Review Length Distribution (After Filtering)



Insight: After filtering, the majority of reviews fall between 10 and 150 words, ensuring text inputs are neither too sparse nor too verbose.

4.5 Review Rating Distribution Using Piechart (Before imbalancing)



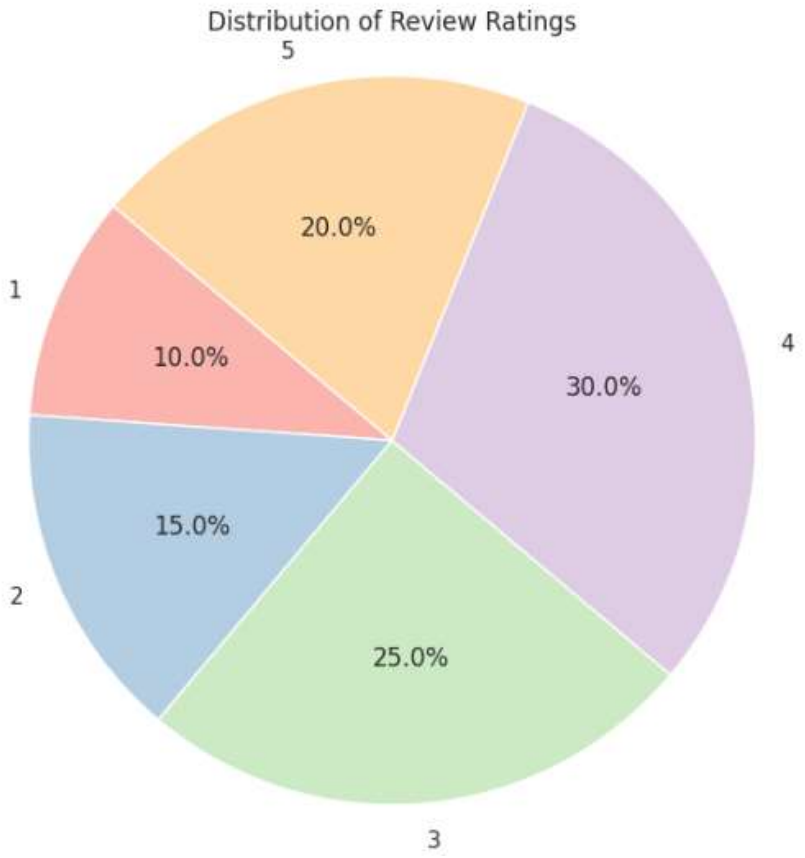
Insight:
The pie chart highlights a **highly imbalanced dataset**, with the majority of reviews skewed toward higher ratings:

- **5-star reviews** dominate at **58.2%** of the total.
- **4-star**: 13.0%
- **3-star**: 11.9%
- **2-star**: 8.4%

- **1-star**: 8.4%

This imbalance indicates that without correction, the model would likely be biased toward predicting 5-star reviews more frequently undermining performance on lower-rated reviews. Such class imbalance can lead to poor generalization and misclassification of negative feedback, which is often the most important to detect in real-world applications.

4.2 Review Rating Distribution Using Piechart (After imbalancing)



Insight:
Insight: This design introduces imbalance deliberately while preserving representation across all rating categories. It helps evaluate how models perform under unequal class conditions without being heavily dominated by any single class.

5. Train-Test Split

To evaluate the model's performance fairly and prevent overfitting, the cleaned and balanced dataset was split into training and testing subsets.

```
# Split training and testing data
X_train, X_test, y_train, y_test = train_test_split(df['reviewText'],df['Rating'], test_size=0.2, random_state=42,  shuffle=True,stratify=df['Rating'] )
```

Split Strategy:

- **Method**: Stratified Train-Test Split
- **Tool**: `train_test_split()` from Scikit-learn
- **Split Ratio**:
  - **80% Training Set**
  - **20% Testing Set**
- **Stratification**: Ensured the proportion of each rating class (1 to 5 stars) remains equal in both training and test sets.

Why Stratification?

Stratified sampling was used to **maintain class balance** in both the training and test sets. Without it, the test set might become imbalanced—even if the original dataset was balanced.

Resulting Dataset Sizes:

Subset	Number of Reviews
Training Set	- 8,000
Testing Set	- 2,000

5.1Post-Split Preprocessing & Vectorization

After splitting the dataset into training and testing sets, additional natural language processing steps were applied to prepare the review text for machine learning.

Stopword Removal & Lemmatization

To improve the semantic quality of the review text, two critical NLP operations were applied:

Code:

```
def spacy_preprocess_pipe(texts):
  processed = []
  for doc in nlp.pipe(texts, batch_size=1000, disable=["ner", "parser"]):  # disable unneeded
parts
    tokens = [
      token.lemma_
      for token in doc
      if not token.is_stop and not token.is_space and token.is_alpha
    ]
    processed.append(" ".join(tokens))
  return processed
```

```
#print the list of stopwords

print(sorted(nlp.Defaults.stop_words))

# Print total number of unique stopwords in spaCy's English model

print("Number of spaCy stopwords:", len(nlp.Defaults.stop_words))
```

```
["'d", "'ll", "'m", "'re", "'s", "'ve", 'a', 'about', 'above', 'across', 'after',
'afterwards', 'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'a
lso', 'although', 'always', 'am', 'among', 'amongst', 'amount', 'an', 'and', 'anot
her', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'anywhere', 'are', 'around'
, 'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becoming',
'been', 'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besides', '
between', 'beyond', 'both', 'bottom', 'but', 'by', 'ca', 'call', 'can', 'cannot',
'could', 'did', 'do', 'does', 'doing', 'done', 'down', 'due', 'during', 'each', 'e
ight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'even', 'ever',
'every', 'everyone', 'everything', 'everywhere', 'except', 'few', 'fifteen', 'fift
y', 'first', 'five', 'for', 'former', 'formerly', 'forty', 'four', 'from', 'front'
, 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'have', 'he', 'hence', 'he
r', 'here', 'hereafter', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him',
'himself', 'his', 'how', 'however', 'hundred', 'i', 'if', 'in', 'indeed', 'into',
'is', 'it', 'its', 'itself', 'just', 'keep', 'last', 'latter', 'latterly', 'least'
, 'less', 'made', 'make', 'many', 'may', 'me', 'meanwhile', 'might', 'mine', 'more
', 'moreover', 'most', 'mostly', 'move', 'much', 'must', 'my', 'myself', "n't", 'n
ame', 'namely', 'neither', 'never', 'nevertheless', 'next', 'nine', 'no', 'nobody'
, 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'n‘t', 'n’t', 'of',
'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or', 'other', 'others', 'oth
erwise', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps
', 'please', 'put', 'quite', 'rather', 're', 'really', 'regarding', 'same', 'say',
'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should'
, 'show', 'side', 'since', 'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'so
mething', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'take', 'ten', 't
han', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'th
ereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'third
', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 't
o', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two', 'und
er', 'unless', 'until', 'up', 'upon', 'us', 'used', 'using', 'various', 'very', 'v
ia', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever'
, 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'wherever',
'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose'
, 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'your
s', 'yourself', 'yourselves', '‘d', '‘ll', '‘m', '‘re', '‘s', '‘ve', '’d', '’ll',
'’m', '’re', '’s', '’ve']

Number of spaCy stopwords: 326
```

X_train = pd.Series(spacy_preprocess_pipe(X_train))

X_test = pd.Series(spacy_preprocess_pipe(X_test))

Stopword Removal:

- **Stopwords** are common words that occur frequently in language (e.g., “the”, “is”, “and”) but carry minimal semantic meaning.

- Removing them helps reduce noise and dimensionality in the vectorized feature space.
- In this project, stopwords were removed using:
    - **spaCy's default English stopword set**, which offers linguistic richness and flexibility.

Lemmatization:

- **Lemmatization** reduces each word to its base or **dictionary form** (called a "lemma").
- Unlike stemming (which can be more aggressive and less accurate), lemmatization ensures that words like:
    - `"running"`, `"ran"` → `"run"`
    - `"better"` → `"good"`
- This step helps group similar words and improves the model's ability to generalize across different grammatical forms.
- Lemmatization was performed using **spaCy**, a fast and powerful NLP library in Python.

About spaCy:

- SpaCy provides **tokenization**, **POS tagging**, **lemmatization**, **NER**, and **syntactic parsing**.
- The English language model used here is:

  ```
  python -m spacy download en_core_web_sm
  ```

  This model provides pre-trained weights for English NLP tasks, including lemmatization.

Note:

Both stopword removal and lemmatization were applied **after splitting** the data, and **separately on the training and testing sets**, to avoid **data leakage**. This ensures the model doesn't "peek" into the test set during training.

5.2 Tokenization

Tokenization is the foundational step in preparing text data for machine learning models. It involves breaking down raw text into smaller units called "tokens", which can be words, subwords, or even characters. This process converts human-readable text into a numerical format that computers can process, enabling further analysis and model training. Tokenization is crucial for tasks like sentiment analysis, text classification, and machine translation, as it helps models to better understand the semantic meaning and context of words within a sentence or document

```
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
# Define tokenizer parameters
vocab_size = 10000
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")

# Fit tokenizer on training data
tokenizer.fit_on_texts(X_train)

# Convert text to sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
```

Text needs to be converted into a format that a deep learning model can understand (numbers). Tokenization is the process of breaking down raw text into smaller units called tokens (typically words or subwords). The `Tokenizer` class from Keras is used for this purpose.

- `texts_to_sequences` transforms each text in `x_test` into a sequence of integers. Each integer represents a specific token (word) based on the vocabulary created by the tokenizer during its initialization and fitting on training data

5.3 Padding

Padding is a technique used in sequence processing to standardize the length of tokenized text sequences. Since sentences and documents naturally vary in length, machine learning models often require uniform input dimensions. Padding addresses this by adding special tokens, typically zeros, to the beginning or end of shorter sequences until they reach a predetermined maximum length. This ensures compatibility with models expecting fixed-size inputs and allows for efficient batch processing during training, while truncation handles longer sequences by removing tokens from either the beginning or end.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len = 100

X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding='post',
truncating='post')
```

```
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding='post', truncating='post')
```

Neural networks often require input sequences to have a uniform length. Since sentences or text segments can vary in length, padding is applied to ensure that all sequences are of the same length by adding a predefined numeric value (usually 0) to shorter sequences.

- `pad_sequences` is a Keras function that pads or truncates sequences to a desired length (`maxlen`).
- `maxlen=max_len`: This parameter sets the maximum length for all sequences. Sequences shorter than `max_len` will be padded, while sequences longer than `max_len` will be truncated.
- `padding='post'`: This specifies that padding (adding zeros) should be done *after* the sequence.
- `truncating='post'`: This indicates that if a sequence is longer than `maxlen`, the truncation (removal of tokens) should happen *from the end* of the sequence.

Deep Learning ModelB