Automated Review Rating System

Intoduction:

In the digital age, customers rely heavily on online reviews before making purchasing decisions. However, many reviews lack explicit ratings or contain subjective sentiments that are difficult to interpret at scale. The goal of this project, **Automated Review Rating System**, is to build a machine learning pipeline that predicts a review's star rating (from 1 to 5) based solely on its textual content.

This system automates the process of rating reviews by:

- Cleaning and preprocessing raw review text.
- Visualizing review patterns and distributions.
- Creating a balanced dataset for fair model training.
- Splitting the data appropriately for training and evaluation.
- Applying vectorization techniques like **TF-IDF** to prepare the text for model input.

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`
- `nltk`
- `spacy`

Install them using:

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

Additional Setup

- Download the English NLP model for spaCy:

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

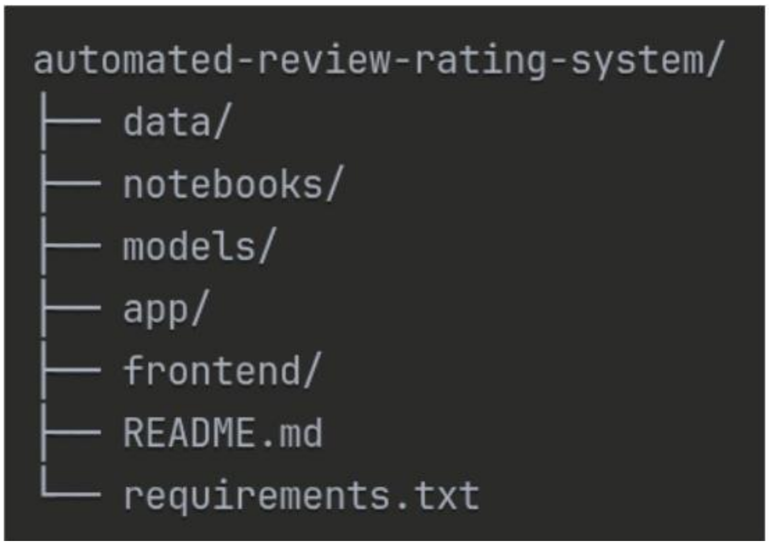
Github Project setup:

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:



```
automated-review-rating-system/
├── data/
├── notebooks/
├── models/
├── app/
├── frontend/
├── README.md
└── requirements.txt
```

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.

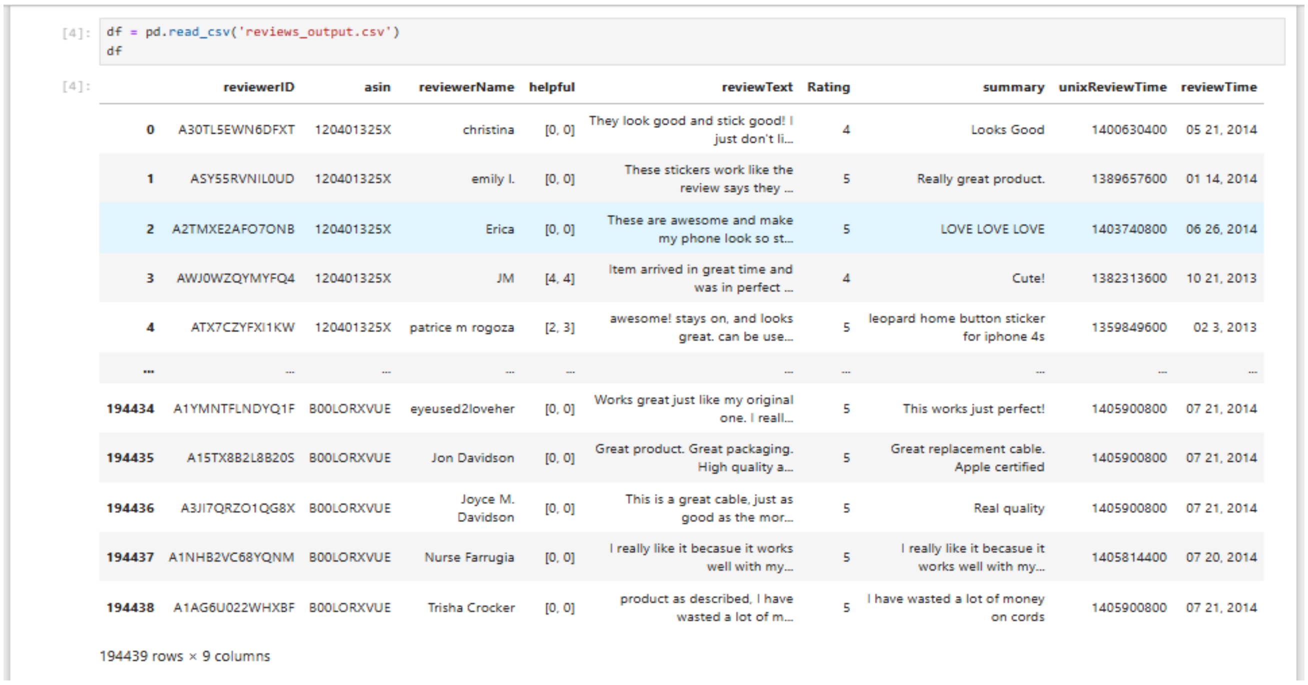
Dataset Overview:

- **Source**: Kaggle - Amazon Reviews: Cell Phones and Accessories
- **Dataset Link: https://github.com/hannafarsin/automatic-review-rating-system/blob/main/data/kaggle/reviews_output.csv**
- **Format**: CSV
- **Size**: 190,000 reviews
- **Key Columns**:
    - `reviewText`: The main review written by the customer.
    - `Rating` (or `overall`): The star rating given by the user (ranging from 1 to 5).
    - Additional fields like `reviewTime`, `reviewerID`, `summary`, etc., were available but not essential for this task.

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:



```
[4]: df = pd.read_csv('reviews_output.csv')
     df
```

[4]:

	reviewerID	asin	reviewerName	helpful	reviewText	Rating	summary	unixReviewTime	reviewTime
0	A30TL5EWN6DFXT	120401325X	christina	[0, 0]	They look good and stick good! I just don't li...	4	Looks Good	1400630400	05 21, 2014
1	ASY55RVNIL0UD	120401325X	emily l.	[0, 0]	These stickers work like the review says they ...	5	Really great product.	1389657600	01 14, 2014
2	A2TMXE2AFO7ONB	120401325X	Erica	[0, 0]	These are awesome and make my phone look so st...	5	LOVE LOVE LOVE	1403740800	06 26, 2014
3	AWJ0WZQYMYFQ4	120401325X	JM	[4, 4]	Item arrived in great time and was in perfect ...	4	Cute!	1382313600	10 21, 2013
4	ATX7CZYFXI1KW	120401325X	patrice m rogoza	[2, 3]	awesome! stays on, and looks great. can be use...	5	leopard home button sticker for iphone 4s	1359849600	02 3, 2013
...	...	...	...	...	...	...	...	...	...
194434	A1YMNTFLNDYQ1F	B00LORXVUE	eyeused2loveher	[0, 0]	Works great just like my original one. I reall...	5	This works just perfect!	1405900800	07 21, 2014
194435	A15TX8B2L8B20S	B00LORXVUE	Jon Davidson	[0, 0]	Great product. Great packaging. High quality a...	5	Great replacement cable. Apple certified	1405900800	07 21, 2014
194436	A3JI7QRZO1QG8X	B00LORXVUE	Joyce M. Davidson	[0, 0]	This is a great cable, just as good as the mor...	5	Real quality	1405900800	07 21, 2014
194437	A1NHB2VC68YQNM	B00LORXVUE	Nurse Farrugia	[0, 0]	I really like it becasue it works well with my...	5	I really like it becasue it works well with my...	1405814400	07 20, 2014
194438	A1AG6U022WHXBF	B00LORXVUE	Trisha Crocker	[0, 0]	product as described, I have wasted a lot of m...	5	I have wasted a lot of money on cords	1405900800	07 21, 2014

194439 rows × 9 columns

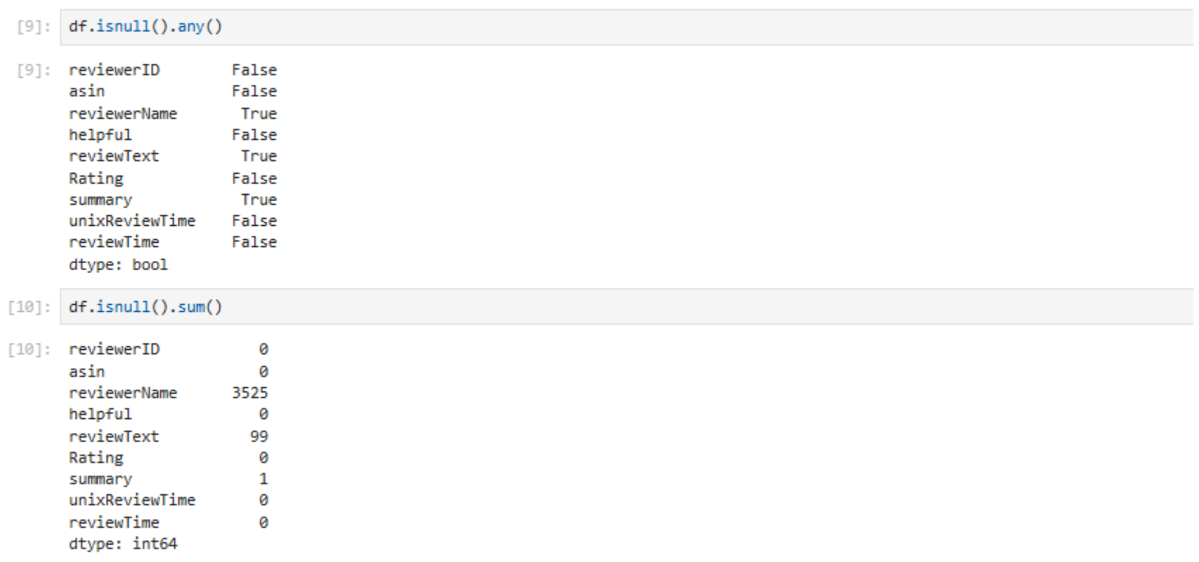
2. Data Preprocessing:

2.1 Initial Data Inspection

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

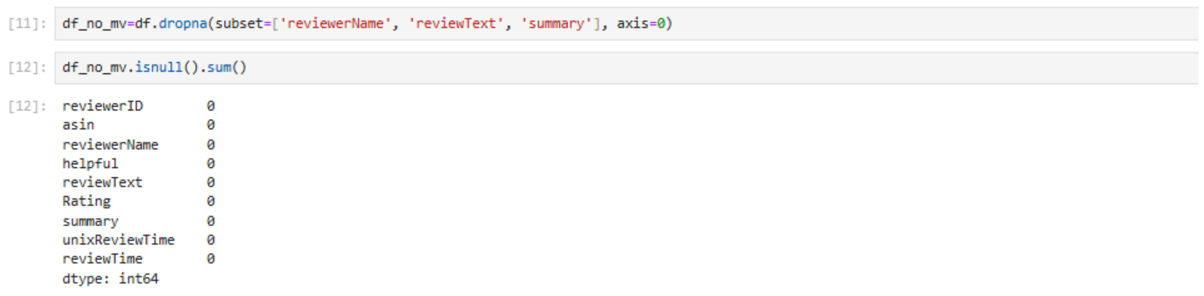
Missing (null) values:

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



```
[9]: df.isnull().any()

[9]: reviewerID        False
     asin              False
     reviewerName       True
     helpful           False
     reviewText         True
     Rating            False
     summary            True
     unixReviewTime    False
     reviewTime        False
     dtype: bool

[10]: df.isnull().sum()

[10]: reviewerID           0
      asin                 0
      reviewerName      3525
      helpful              0
      reviewText          99
      Rating               0
      summary              1
      unixReviewTime       0
      reviewTime           0
      dtype: int64
```

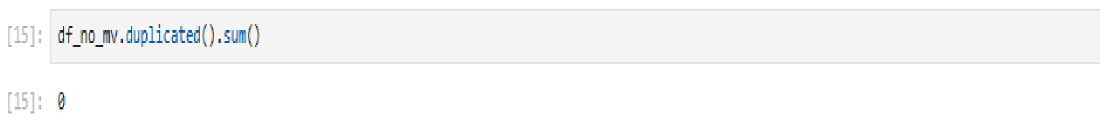
Rows with missing `reviewText` or `Rating` were removed.



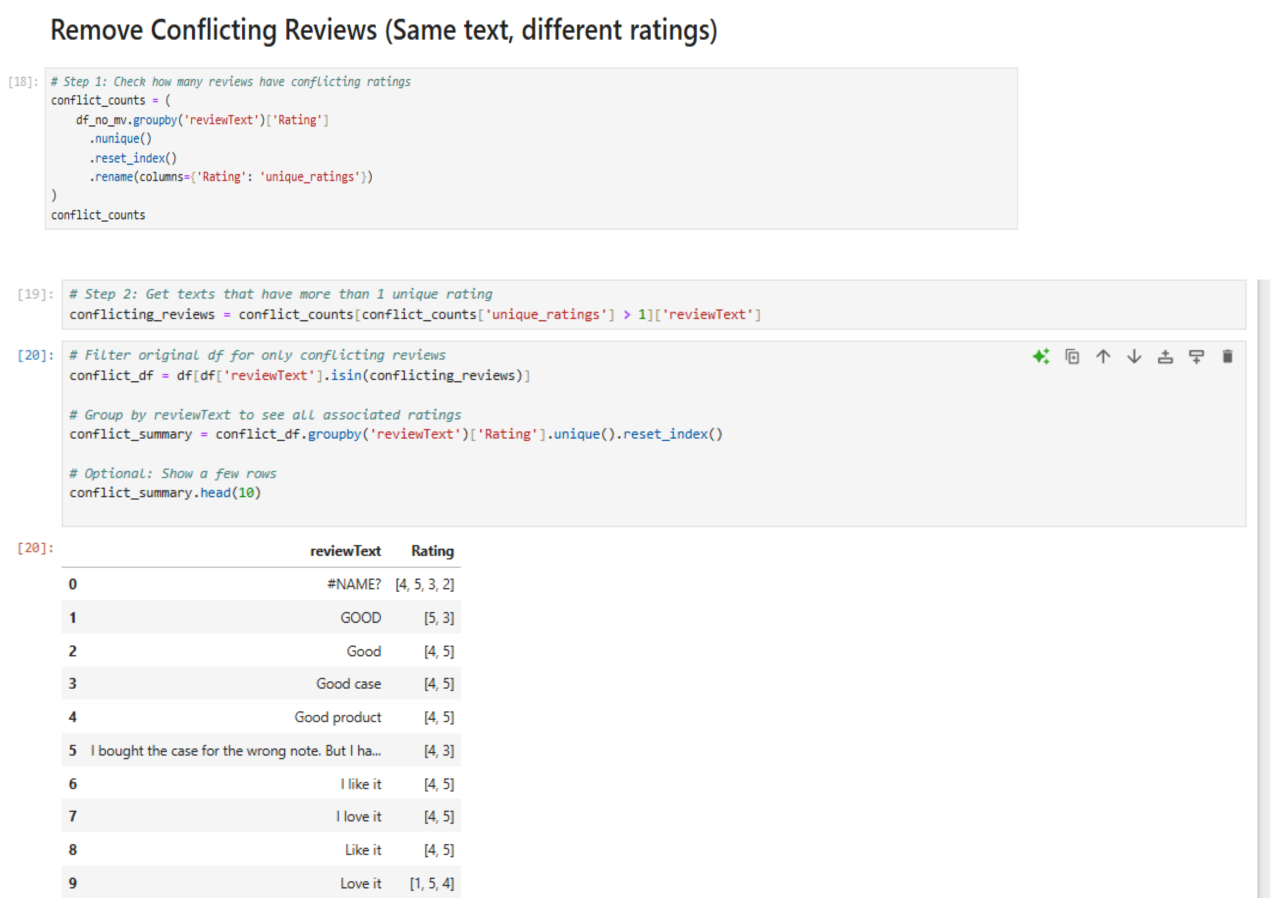
```
[11]: df_no_mv=df.dropna(subset=['reviewerName', 'reviewText', 'summary'], axis=0)

[12]: df_no_mv.isnull().sum()

[12]: reviewerID        0
      asin              0
      reviewerName      0
      helpful           0
      reviewText        0
      Rating            0
      summary           0
      unixReviewTime    0
      reviewTime        0
      dtype: int64
```

Duplicate entries:

Exact duplicates were identified and dropped to avoid model bias.



```
[15]: df_no_mv.duplicated().sum()

[15]: 0
```



Remove Conflicting Reviews (Same text, different ratings)

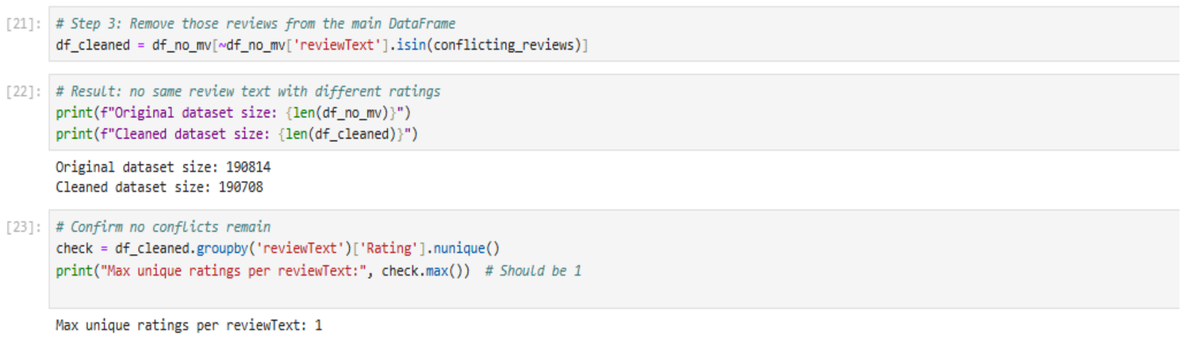
```
[18]: # Step 1: Check how many reviews have conflicting ratings
      conflict_counts = (
          df_no_mv.groupby('reviewText')['Rating']
            .nunique()
            .reset_index()
            .rename(columns={'Rating': 'unique_ratings'})
      )
      conflict_counts
```

```
[19]: # Step 2: Get texts that have more than 1 unique rating
      conflicting_reviews = conflict_counts[conflict_counts['unique_ratings'] > 1]['reviewText']
```

```
[20]: # Filter original df for only conflicting reviews
      conflict_df = df[df['reviewText'].isin(conflicting_reviews)]

      # Group by reviewText to see all associated ratings
      conflict_summary = conflict_df.groupby('reviewText')['Rating'].unique().reset_index()

      # Optional: Show a few rows
      conflict_summary.head(10)
```

[20]:

	reviewText	Rating
0	#NAME?	[4, 5, 3, 2]
1	GOOD	[5, 3]
2	Good	[4, 5]
3	Good case	[4, 5]
4	Good product	[4, 5]
5	I bought the case for the wrong note. But I ha...	[4, 3]
6	I like it	[4, 5]
7	I love it	[4, 5]
8	Like it	[4, 5]
9	Love it	[1, 5, 4]



```
[21]: # Step 3: Remove those reviews from the main DataFrame
      df_cleaned = df_no_mv[~df_no_mv['reviewText'].isin(conflicting_reviews)]
```

```
[22]: # Result: no same review text with different ratings
      print(f"Original dataset size: {len(df_no_mv)}")
      print(f"Cleaned dataset size: {len(df_cleaned)}")

      Original dataset size: 190814
      Cleaned dataset size: 190708
```

```
[23]: # Confirm no conflicts remain
      check = df_cleaned.groupby('reviewText')['Rating'].nunique()
      print("Max unique ratings per reviewText:", check.max())  # Should be 1

      Max unique ratings per reviewText: 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.

```
[21]: # Step 3: Remove those reviews from the main DataFrame
      df_cleaned = df_no_mv[~df_no_mv['reviewText'].isin(conflicting_reviews)]
```

```
[22]: # Result: no same review text with different ratings
      print(f"Original dataset size: {len(df_no_mv)}")
      print(f"Cleaned dataset size: {len(df_cleaned)}")

      Original dataset size: 190814
      Cleaned dataset size: 190708
```

```
[23]: # Confirm no conflicts remain
      check = df_cleaned.groupby('reviewText')['Rating'].nunique()
      print("Max unique ratings per reviewText:", check.max())  # Should be 1

      Max unique ratings per reviewText: 1
```

Cleaning Steps Applied:

1. **Lowercasing**

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

2. **Removing URLs and 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

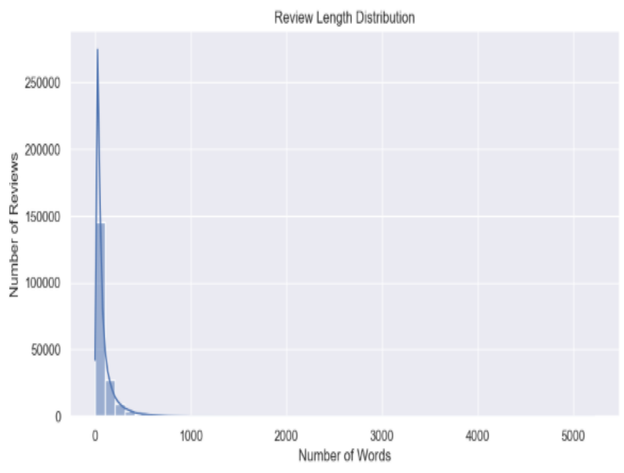
- Emojis, symbols, and non-ASCII characters were stripped from the text.
- Example:
  `"Great phone 😍✨!"` → `"great phone"`

4. Removing Punctuation and Numbers

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

5. **Review Length 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.



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 100 words**, with a sharp peak under 50 words.
- A **long tail of outliers** includes reviews with extremely high word counts, some exceeding **5000 words**.

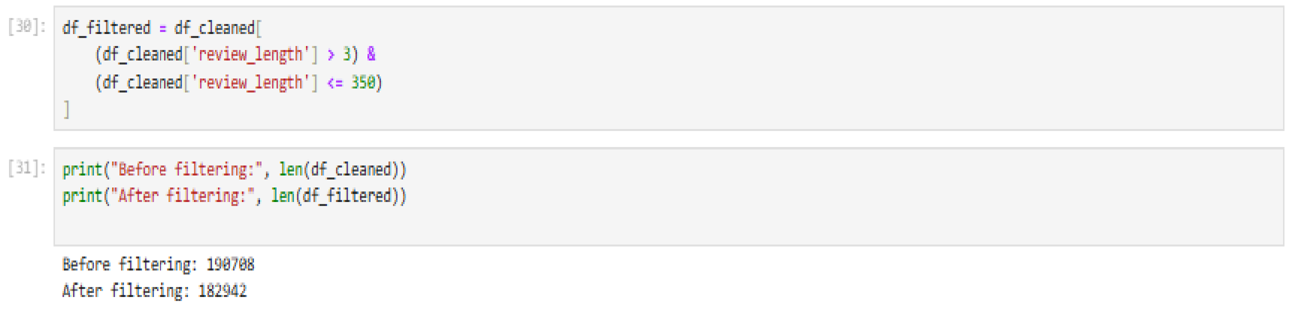
To improve data quality and processing efficiency:

- **Very short reviews** (less than 3 words) were removed, as they offer limited semantic information.
- **Excessively long reviews** (e.g., over 350–500 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:

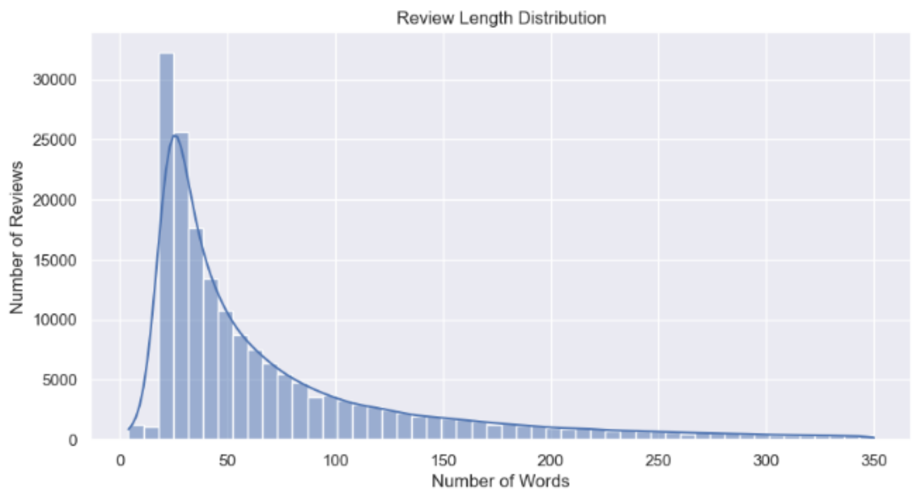
- Very short reviews (fewer than 3 words) and excessively long ones (e.g., over 350 words) were removed.
- This helped eliminate outliers and non-informative entries



```
[30]: df_filtered = df_cleaned[
          (df_cleaned['review_length'] > 3) &
          (df_cleaned['review_length'] <= 350)
      ]
```

```
[31]: print("Before filtering:", len(df_cleaned))
      print("After filtering:", len(df_filtered))
```

```
Before filtering: 190708
After filtering: 182942
```



After applying the text cleaning and length-based filtering:

- Reviews with **fewer than 3 words** and those with **excessive lengths (over 350 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.

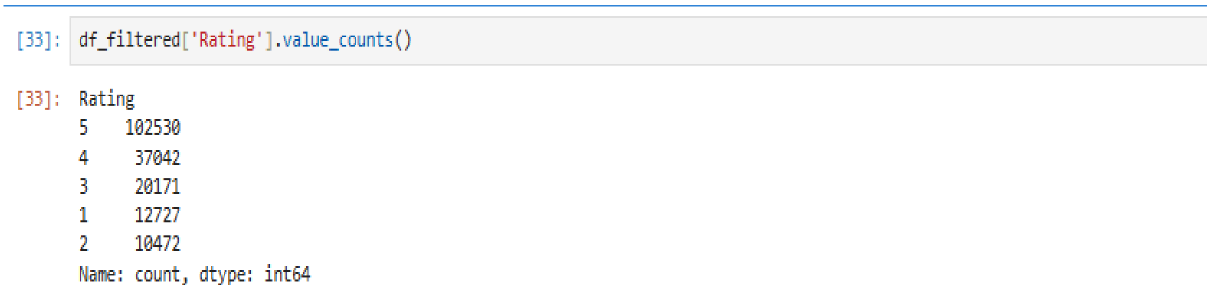
3. Creating a Balanced Dataset

Why Balance the Dataset?

The original dataset was **heavily imbalanced**, with a significantly higher number of 5-star reviews compared to lower ratings. This can lead to serious problems during model training, such as:

- ❗ The model becomes **biased toward majority classes** (e.g., 5-star).
- ❌ **Poor generalization** on minority classes like 1- or 2-star reviews.
- 🎯 **Decreased accuracy and fairness**, especially in real-world scenarios where predicting low ratings is crucial for quality control.

Rating Distribution across 1 through 5 stars are:



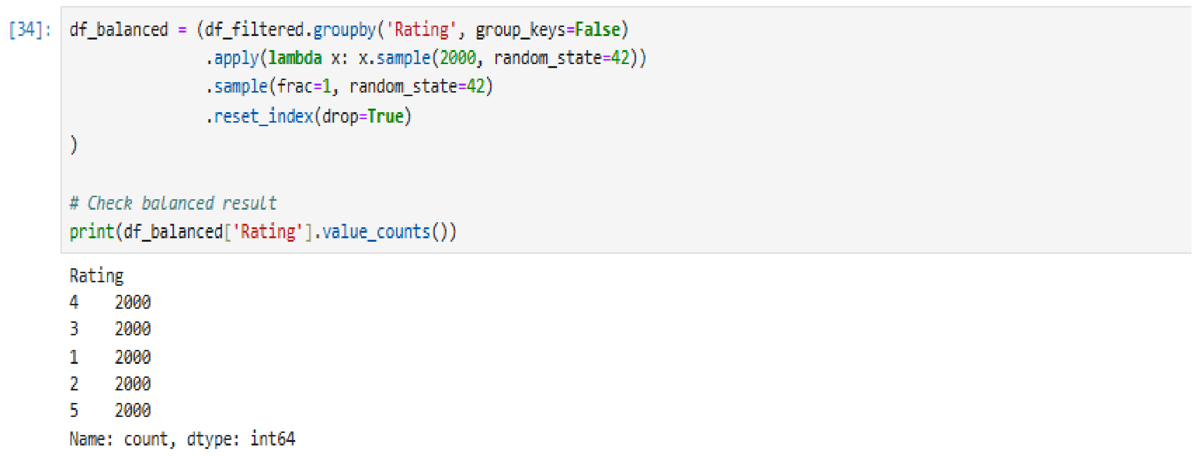
```
[33]: df_filtered['Rating'].value_counts()

[33]: Rating
      5    102530
      4     37042
      3     20171
      1     12727
      2     10472
      Name: count, dtype: int64
```

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 would cause the model to **rarely predict low ratings**, reducing its usefulness.

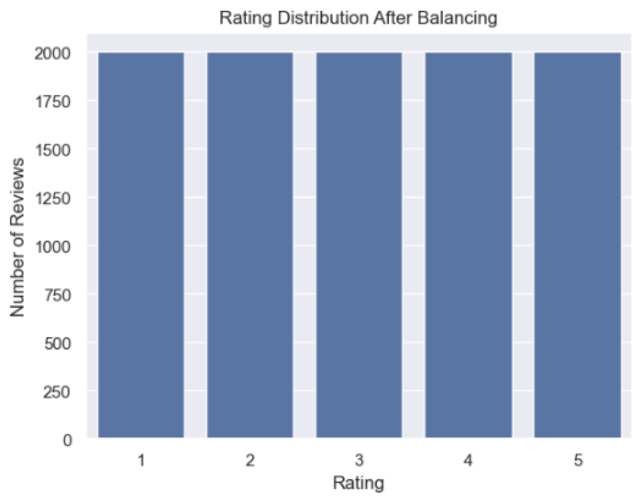
Balancing Strategy:

To address this, the dataset was balanced by:

- Sampling **2,000 reviews per class (1–5 stars)**.
- This resulted in a **balanced dataset of 10,000 reviews** with equal representation from each class.



```
[34]: df_balanced = (df_filtered.groupby('Rating', group_keys=False)
                     .apply(lambda x: x.sample(2000, random_state=42))
                     .sample(frac=1, random_state=42)
                     .reset_index(drop=True)
      )

      # Check balanced result
      print(df_balanced['Rating'].value_counts())

      Rating
      4    2000
      3    2000
      1    2000
      2    2000
      5    2000
      Name: count, dtype: int64
```



Balanced 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 **2,000 reviews**, ensuring equal representation across all categories.

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

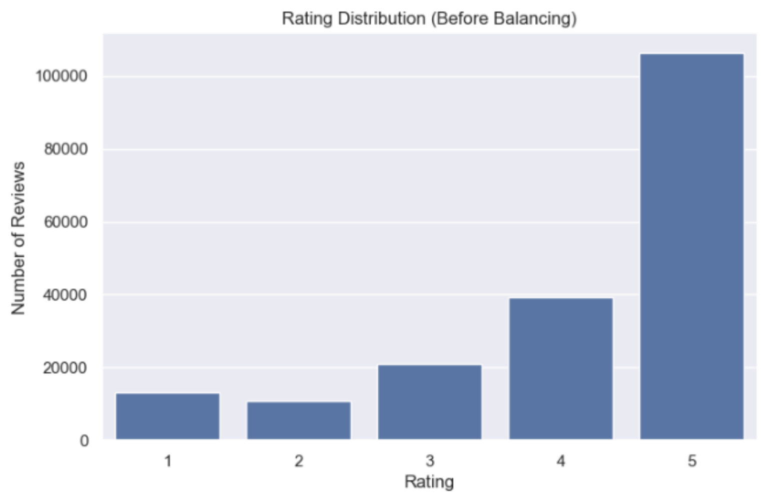
- Prevent model bias toward majority classes
- Improve accuracy for underrepresented ratings
- Ensure fair and consistent performance during training and evaluation

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

4. Data Visualization

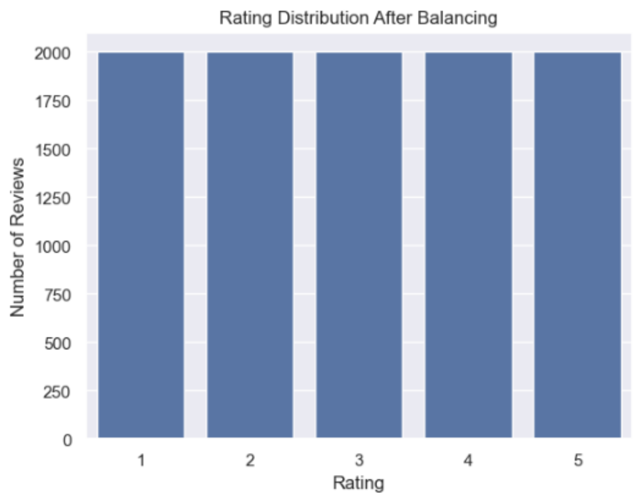
To better understand the characteristics of the reviews and guide preprocessing decisions, several visualizations were created. These plots offer insights into rating distribution, review length, and review content.

4.1 Review Rating Distribution (Before Balancing)



Insight: The original dataset was highly imbalanced, with over 100,000 reviews rated 5 stars, while 1- and 2-star reviews were much less frequent. This confirmed the need for dataset balancing before training.

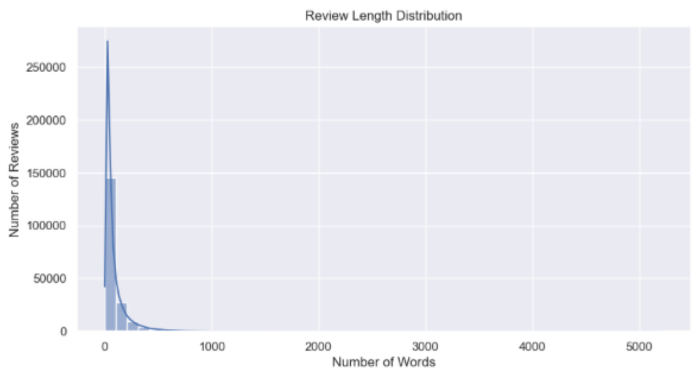
4.2 Review Rating Distribution (After Balancing)



Insight:
The distribution is now perfectly uniform, with **2,000 reviews for each rating class (1 to 5 stars)**. This ensures that the machine learning model will be trained on an **equal representation of all sentiment levels**, which helps:

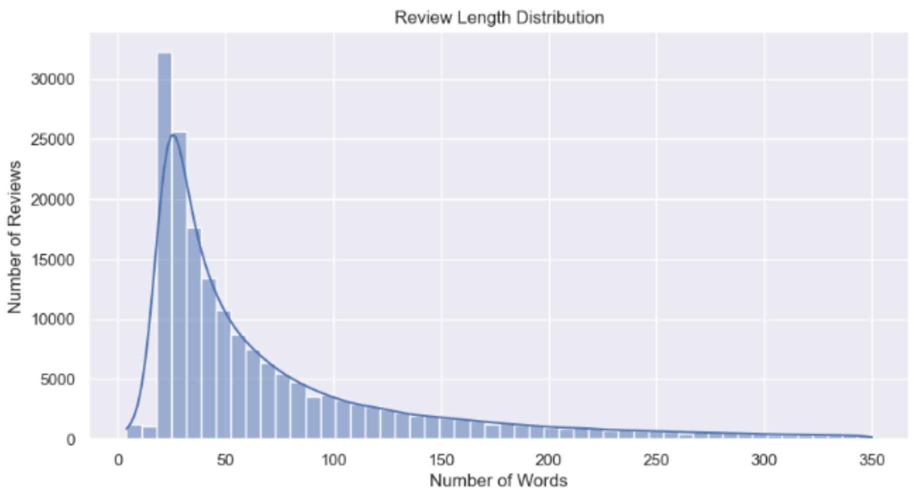
- Prevent overfitting to dominant classes (like 5-star reviews),
- Improve prediction accuracy for underrepresented ratings (1-star and 2-star),
- Enhance the **fairness and robustness** of the rating prediction system.

4.3 Review Length Distribution (Before Filtering)



Insight: Most reviews are under 100 words, but some extend beyond 5,000 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.

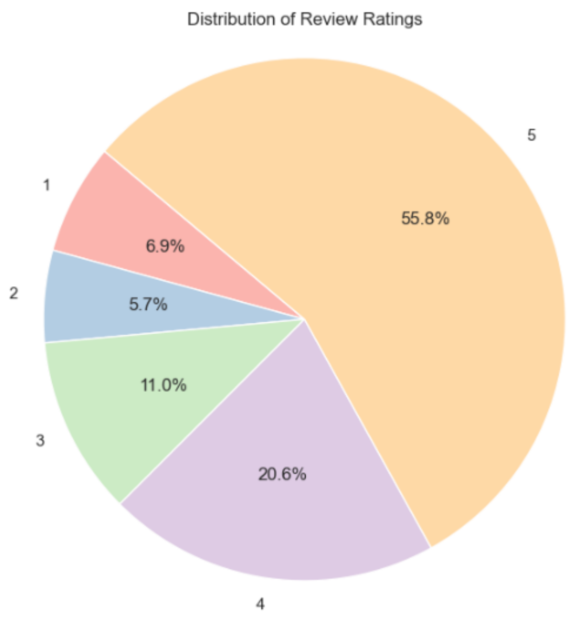
4.5 Review Rating Distribution Using Piechart (Before Balancing)

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

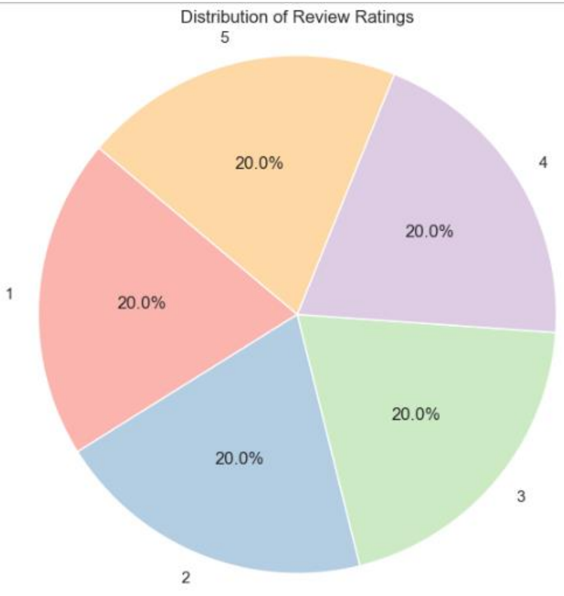
- **5-star reviews** dominate at **55.8%** of the total.
- **4-star**: 20.6%

- **3-star**: 11.0%
- **2-star**: 5.7%
- **1-star**: 6.9%

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 Balancing)



Insight:
The pie chart shows that the dataset has been successfully balanced, with **each rating class (1**

to 5 stars) now contributing **exactly 20%** of the total reviews. This uniform distribution ensures:

- Equal learning opportunity for each class
- Fairer model training with **no class dominating the prediction output**
- Better performance on previously underrepresented ratings (especially 1-star and 2-star)

Balancing the dataset at this stage helps reduce bias, improve classification accuracy, and support robust model generalization across all rating levels.

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 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:

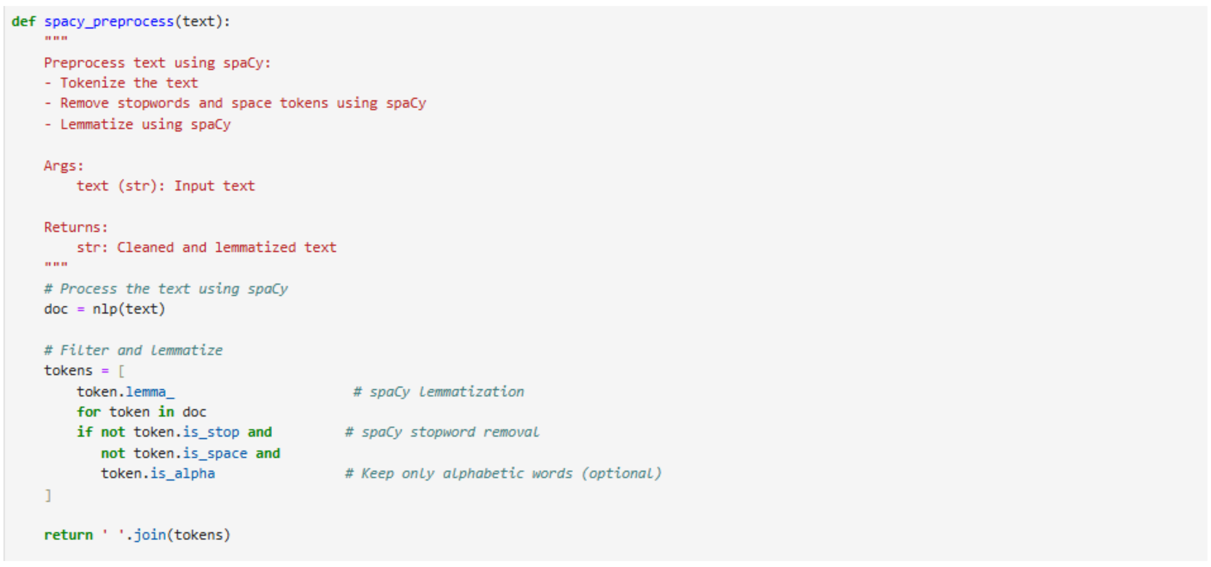
Subset	Number of Reviews
Training Set	- 8,000
Testing Set	- 2,000

5.1Post-Split Preprocessing & Vectorization

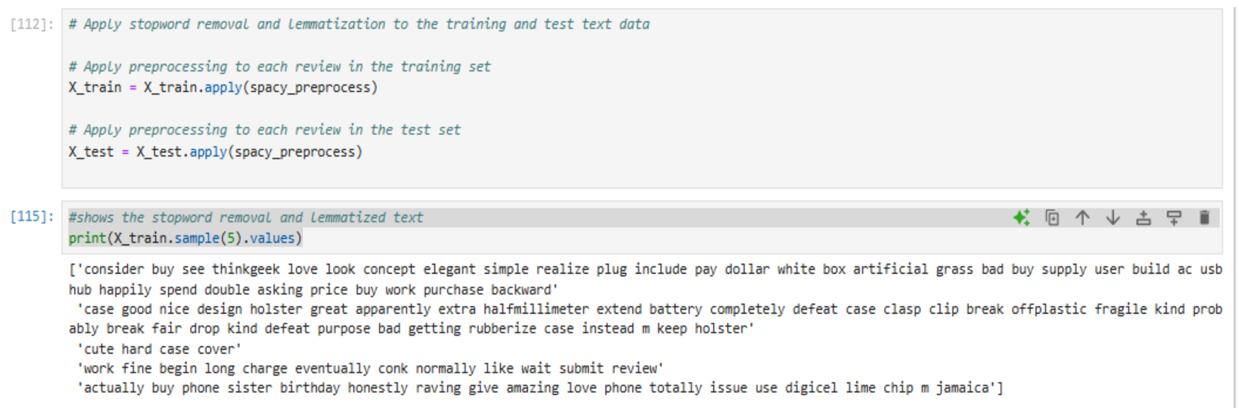
After splitting the dataset into training and testing sets, additional natural language processing steps were applied to prepare the review text for machine learning.

Stopword Removal & Lemmatization

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



```python
def spacy_preprocess(text):
    """
    Preprocess text using spaCy:
    - Tokenize the text
    - Remove stopwords and space tokens using spaCy
    - Lemmatize using spaCy

    Args:
        text (str): Input text

    Returns:
        str: Cleaned and lemmatized text
    """
    # Process the text using spaCy
    doc = nlp(text)

    # Filter and lemmatize
    tokens = [
        token.lemma_                      # spaCy lemmatization
        for token in doc
        if not token.is_stop and         # spaCy stopword removal
           not token.is_space and
           token.is_alpha                # Keep only alphabetic words (optional)
    ]

    return ' '.join(tokens)
```



```
[112]: # Apply stopword removal and lemmatization to the training and test text data

       # Apply preprocessing to each review in the training set
       X_train = X_train.apply(spacy_preprocess)

       # Apply preprocessing to each review in the test set
       X_test = X_test.apply(spacy_preprocess)

[115]: #shows the stopword removal and lemmatized text
       print(X_train.sample(5).values)

       ['consider buy see thinkgeek love look concept elegant simple realize plug include pay dollar white box artificial grass bad buy supply user build ac usb
       hub happily spend double asking price buy work purchase backward'
        'case good nice design holster great apparently extra halfmillimeter extend battery completely defeat case clasp clip break offplastic fragile kind prob
       ably break fair drop kind defeat purpose bad getting rubberize case instead m keep holster'
        'cute hard case cover'
        'work fine begin long charge eventually conk normally like wait submit review'
        'actually buy phone sister birthday honestly raving give amazing love phone totally issue use digicel lime chip m jamaica']
```

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
  ```

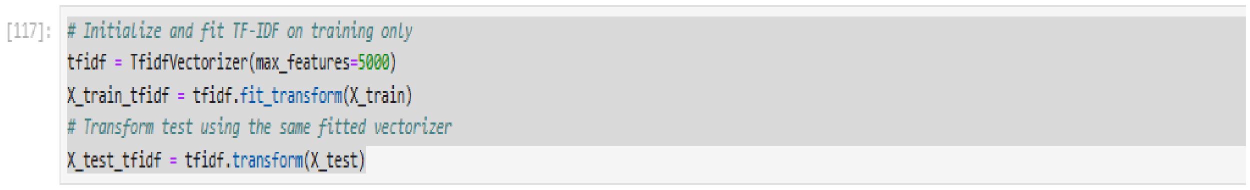
  This model provides pre-trained weights for English NLP tasks, including lemmatization.

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 Text Vectorization (TF-IDF)

Once the text was cleaned and normalized, it needed to be converted into numerical form to be usable by machine learning models. For this, the **TF-IDF (Term Frequency–Inverse Document Frequency)** method was applied.



```
[117]: # Initialize and fit TF-IDF on training only
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
# Transform test using the same fitted vectorizer
X_test_tfidf = tfidf.transform(X_test)
```

What is TF-IDF?

- TF-IDF transforms each review into a numerical vector based on:
  - **Term Frequency (TF)**: How often a word appears in a review.
  - **Inverse Document Frequency (IDF)**: How unique or rare a word is across the entire dataset.
- This gives higher importance to words that are **frequent in a document but rare overall**, improving the model’s focus on distinctive terms.

Vectorization Workflow in the Project:

1. The `TfidfVectorizer` was **fitted only on the training data (`X_train`)** to learn the vocabulary and document frequencies.

2. The **same vectorizer** was then used to transform both:
   - `X_train` (fit and transform)
   - `X_test` (transform only)

Why Not Fit on Test Data?

- Fitting the vectorizer only on training data **prevents data leakage**.
- It simulates a real-world scenario where the model is trained on past data and evaluated on unseen data.

Final Output Variables:

Variable	Description
`X_train`	TF-IDF vectorized training data
`X_test`	TF-IDF vectorized testing data
`y_train`	Star ratings (1–5) for training reviews
`y_test`	Star ratings (1–5) for testing reviews

This final processed dataset is now ready for **model training and evaluation**.