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
- spacy

Install them using:

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

Additional Setup

- Download the English NLP model for spaCy:

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

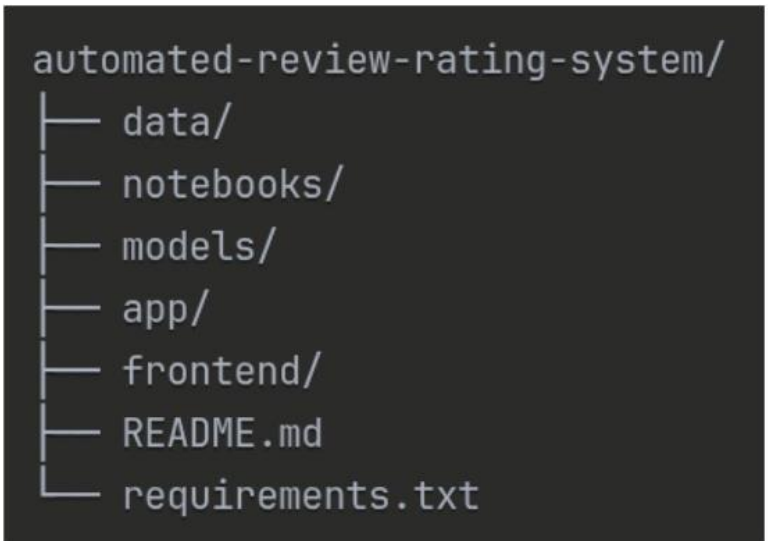
Github Project setup:

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:



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**
- **Balanced Dataset Link:** **https://github.com/hannafarsin/automatic-review-rating-system/blob/main/data/kaggle/balanced_reviews_dataset3.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.

df.isnull().any()

```
reviewerID        False
asin              False
reviewerName       True
helpful           False
reviewText         True
Rating            False
summary            True
unixReviewTime    False
reviewTime        False
dtype: bool
```

df.isnull().sum()

```
reviewerID           0
asin                 0
reviewerName      3525
helpful              0
```

```
reviewText          99
Rating               0
summary              1
unixReviewTime       0
reviewTime           0
dtype: int64
```

`reviewText` or `Rating` were removed.

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

```
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.

```
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('reviewText')['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]['reviewText']
# 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()
#  Show a few rows
conflict_summary.head(10)
```

:

	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]

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['reviewText'] = df_cleaned['reviewText'].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"`

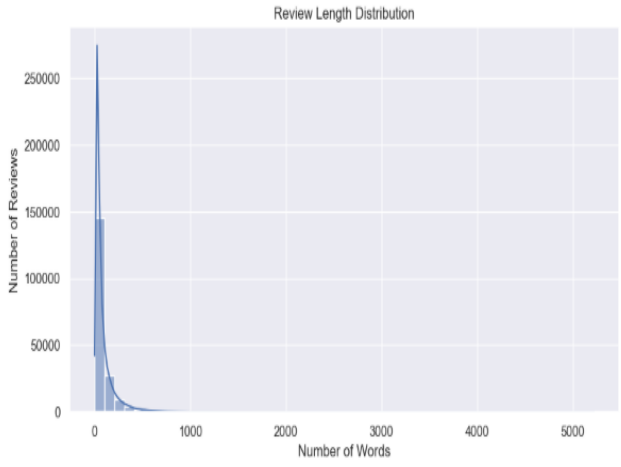
4. Removing Punctuation and Numbers

```
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. **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.

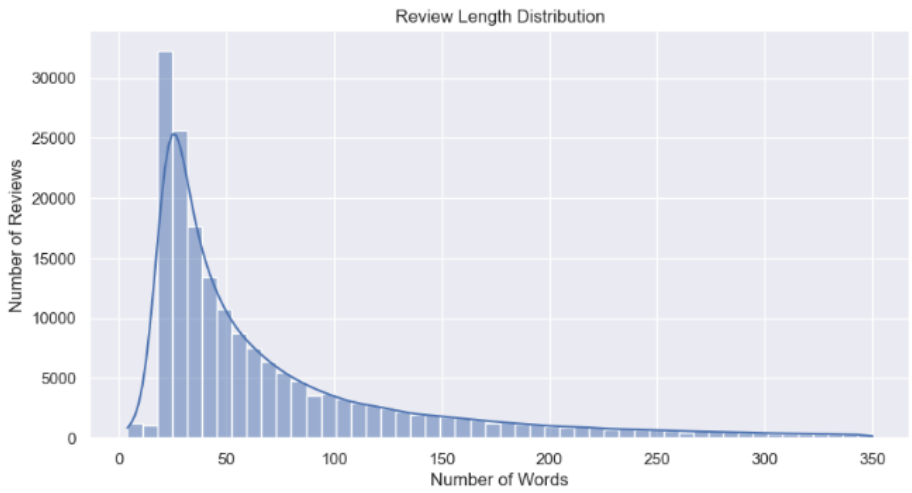
This step ensures a more **uniform distribution**, making the dataset more suitable for vectorization and model training.

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

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

df_filtered['Rating'].value_counts()

```
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.

```
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())
```

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.

Display the statistics of the word count after balancing:

```
df['word_count'] = df['reviewText'].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       2000.0  66.5735  57.399486  4.0  27.0  46.0   82.0  348.0
2       2000.0  79.1530  68.230281  4.0  31.0  53.5  102.0  348.0
3       2000.0  79.1330  71.235585  4.0  29.0  50.0  104.0  350.0
4       2000.0  82.5520  73.502752  4.0  30.0  54.0  106.0  349.0
5       2000.0  66.9570  64.794122  4.0  26.0  40.5   77.0  347.0
```

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['reviewText']}")
```

```
☆ Rating 1
- 3000mah is roughly twice the capacity of an iphone 4 which i used for music and
nike app while trail running the phone alone 1400mah could last 34 hours with audi
o gps and miscellaneous nearconstant activity it stands to reason that the anker 3
000mah could at least double the usable span if not triple itafter a particular ru
n i ran the phone down to 45 and plugged it into the anker while i entertained mys
elf on the phone and waited for a long series of photographic exposures after an h
our of charging the phone the anker had been drained and the phone was back on its
own power without having been recharged further than another 15 by the ankerive go
t another battery cell thats at least four times heavier higher capacity also that
doesnt selfdischarge nearly as badly as this but its half again as heavy as a load
ed ultralight trail running backpack


- this is not the same useful juice pack i have come to expect from mophie saw it
at the apple store today and bought it with out a second thought i got as far as p
lacing it in the case which was awkward to begin with then realized i needed a spe
cial adapter to use my headphone then realized it uses a micro usb not a lightning
```

adapter which i have set up on my desk already for traveling the usb issue is not a huge deal but more of an annoyance but i use a wired headset frequently and having to use an extra adapter does not work if the other items were not a problem for you consider the fact itunes does not recognize the device when plugged in

- a compete waste of money and expectations too small for apple category products and untrustworthy a completely meaningless item and shouldnt be admitted not amazon servers

☆ Rating 2

- was using this as a stand for when i do facetime on my phone it broke when i bent it back too far only 12 an inch or so it uses a small spring to keep the clamp shut the spring is really weak if you bend it back to far i t will not work anymore

- maybe it just didnt fully fit my air duct fully but it just didnt stay put to well and it felt super cheaply made

- i was given an opportunity to review this item through the vine program and after having a chance to try it i think its was too expensive if you plan on using it to change your phone when you are travelingyou will need to purchase duracell powermat access case wireless charging case for iphone 5 to be able to charge your phone wirelesslyenough said why it isnt included at the already seemingly overinflated price im not sureif i need to change my phone quickly and not near an outlet id rather sit in my car and use my car charger rather than spend money for an additional attachment for this productnot recommended

☆ Rating 3

- for the price this bumper is pretty good its definitely not the best but its far from the worst all of the cutouts match perfectly so thats a plus it is a little flimsy but i cant really complain

- if you are looking for a cheap case to protect your screen this is the one for that and that only

- these are the exact ones that i was looking for but there is one problem they are supposed to be clear but they arrived yellowed in the package it is obvious that they are new but they have yellowed and are no longer clear i bought these to replace existing ones that i have that have yellowed but not to the extent of these new ones i have had the previous ones since 2010 i will keep as spares but am unhappy with the condition of these eargels

☆ Rating 4

- these are great quality headphones i love the builtin forward and pause buttons good sound overall for oem you wont be disappointed with these

- ive used this bluetooth for close to three years it worked very well for a long time but recently the sound quality is beginning to wane making it hard to hear it fades in and out from moderately good sound to really poor sound

- good price fast shippingi bought sixone for car office home office three for kid sfits a little tight at the iphone or ipad connector but they work and charge the device just finewe shall see how long they last but i would recommendavailable in multiple colors is a plus too

☆ Rating 5

- this case is now my goto case i use it as my daily for work construction it is rugged and durable i use to use a maxboost shell holster combo because it was also very durable i still like it but this tudia is better because it has a slimmer profile and its bumper border is very rigid it absorbs the drop not the phone the case does have flexibility but the bumper is very strong in fact watch my video on this case on my youtube channel mikepado

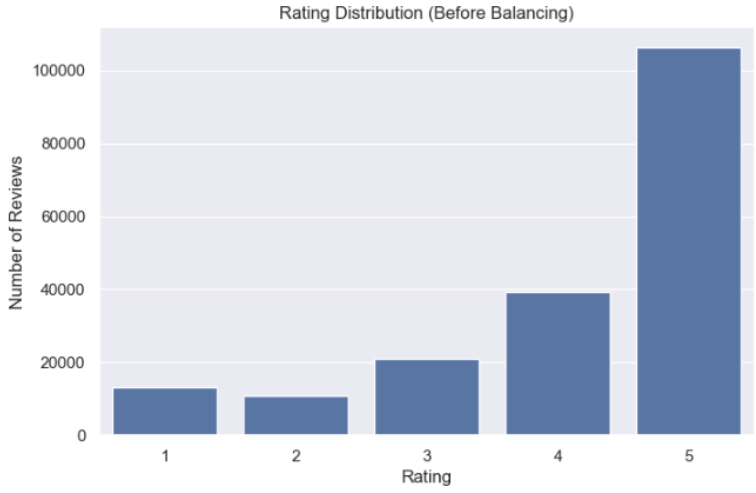
- my husband grew tired of having to replace the otter box for his iphone 4 because the otter box stretches and warps he read about this one online and it has been a great case no complaints and no damage to the phone including a few drops

- all my friends that have a galaxy s 2 wants a cover like this i guess i will be order more for xmas gifts

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)



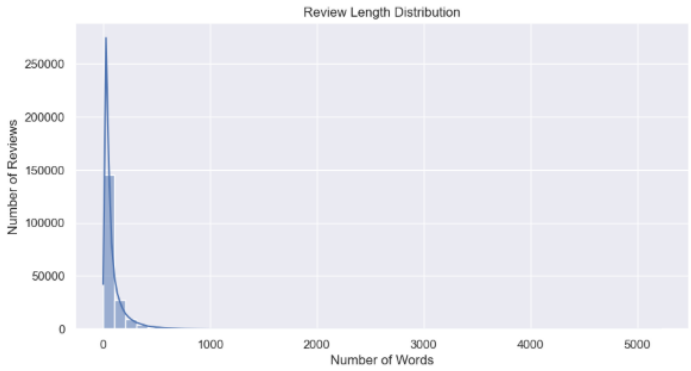
Insight: The original dataset was highly imbalanced, with over 100,000 reviews rated 5 stars, while 1- and 2-star reviews were much less frequent. This confirmed the need for dataset balancing before training.

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:

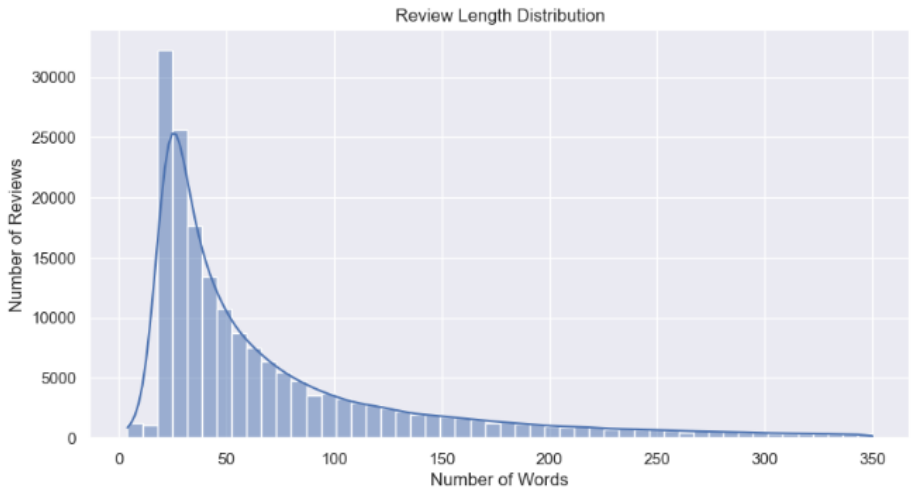
- Prevent overfitting to dominant classes (like 5-star reviews),
- Improve prediction accuracy for underrepresented ratings (1-star and 2-star),
- Enhance the **fairness and robustness** of the rating prediction system.

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)



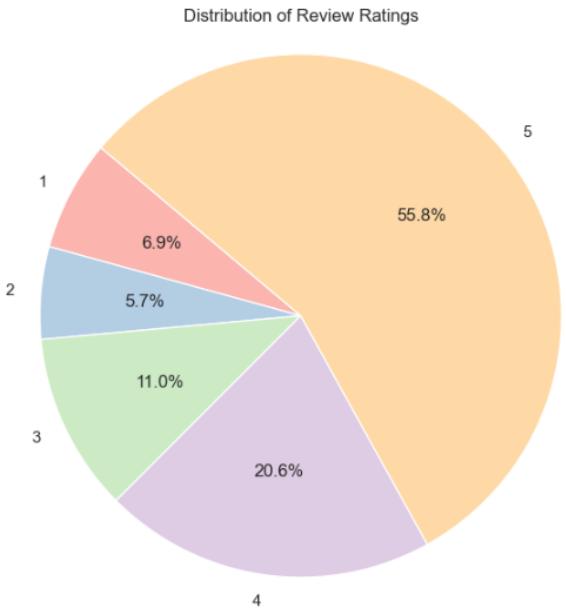
Insight: After filtering, the majority of reviews fall between 10 and 150 words, ensuring text inputs are neither too sparse nor too verbose.

4.5 Review Rating Distribution Using Piechart (Before Balancing)

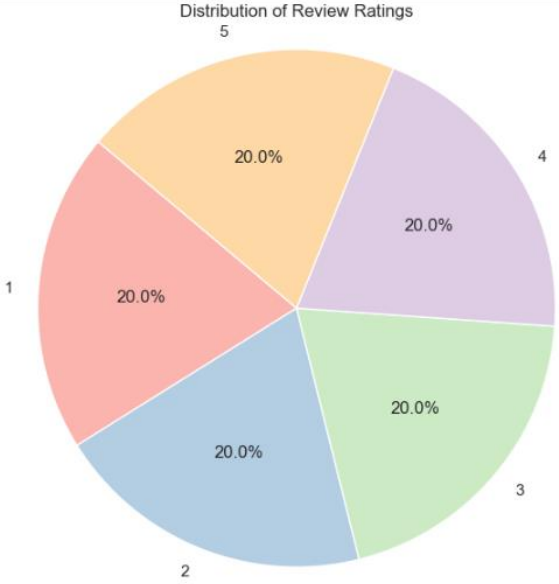
Insight:
The pie chart highlights a **highly imbalanced dataset**, with the majority of reviews skewed toward higher ratings:

- **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 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,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:

```
def spacy_preprocess(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)
#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
```

```
# 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)
```

shows the stopword removal and lemmatized text:

```
['piece conjunction mount truck recently upgrade phone note large phone fit holder
not rest button'

 'base great review accuracy stylus ve decide try stylus thin comfortable hold sec
ond limit certain angel make light screechy sound write ipad screen tip stylus met
al like screechy sound blackboard fourth price stylus pretty steep consider issue
finally writing experience tiresome awkwardness stylus metal tip tap screen naviga
te ipad weird noisy yes easy write benefit marginal weigh con small benefit totall
y weight negative want like stylus point long screechy noise disappear d matter sc
reen sound will not away well styli fraction price stylus try different styli find
one mesh tip styli good accurate smooth smooth effortless write ipad look high rev
iew will not disappoint'

 'take trip new york girl grateful work need little charge small compact'

 'genuine samsung ac travel adapter identical come samsung note phone buy charger
home work catch weak battery need'

 'look great not use flash camera cause come pink hassle']
```

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.

```
# 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?

Term frequency-inverse document frequency (TF-IDF) is a metric you can use in machine learning to create a numerical representation of the words in a text document that demonstrates how relevant those words are within the text as a whole. TF-IDF is an important component of information retrievel and natural language processing because it can help AI models

analyze the keywords and phrases within documents, providing a more nuanced understanding of what the text says.

Term frequency refers to how often a term is used in a body of text. Inverse document frequency looks at how many documents in a body of documents contain that term. To use a metaphor, term frequency considers how many times the word appears in the book, and inverse document frequency looks at how many books in the library use that term. In your local library, nearly every book would use words like “the” or “and” frequently. These words would have both a high term frequency (TF) and inverse document frequency (IDF), so an AI model would understand that those words weren’t particularly helpful for understanding what the document is about.

However, a word like “skydiving” would not appear in as many books. An AI model could use TF-IDF to determine that “skydiving” is a more important word for deciding what a document is about. Books with a higher TF for “skydiving” would be more relevant for people who want to check out a book about preparing for a skydive.

What is TF-IDF used for?

You can use TF-IDF for three primary purposes: information retrieval, keyword extraction, and machine learning.

- **Information retrieval:** TF-IDF allows an AI model to retrieve information from a massive text library. For example, a search engine uses TF-IDF to determine which documents to provide as the result of a search query.
- **Keyword of feature extraction**: TF-IDF provides a mechanism for an AI model to determine the most important words in a text, which the model can then use to summarize the document.
- **Machine learning:** TF-IDF is an important part of natural language processing, a machine learning technique that allows computers and AI models to interpret human language. TF-IDF is important for

training models to understand the patterns behind human language and the importance of individual words.

Calculate term frequency and inverse document frequency

If you wanted to calculate TF-IDF, you would first need to calculate term frequency, then inverse document frequency, and finally TF-IDF. The equations for these calculations include three variables:

- t: term
- d: document
- D: set of documents

The equation to find term frequency is

= **tf(t,d)=log(1+freq(t,d))**

The equation to find inverse document frequency
= **idf(t,D)=log(N/count(d∈D:t∈d))**.

To put them together and find TF-IDF, the equation
= **tfidf(t,d,D)=tf(t,d).idf(t,D).**

Your final results will measure how important the term is within the entire body of documents. The higher the score, the more relevant the AI model will consider that term.

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**.

Machine Learning Algorithms