AUTOMATED RATING REVIEW SYSTEM

Imbalanced Dataset Creation

1.PROJECT OVERVIEW

This project predicts customer review ratings based on textual feedback. It involves cleaning and preprocessing review text and creating an imbalanced dataset by using the remaining reviews not included in the balanced dataset. The dataset is transformed into TF-IDF features, and a stratified train-test split is applied to maintain realistic class distribution. The system helps analyze model performance on skewed data, providing insights into review patterns, customer sentiment, and how models handle imbalanced classes

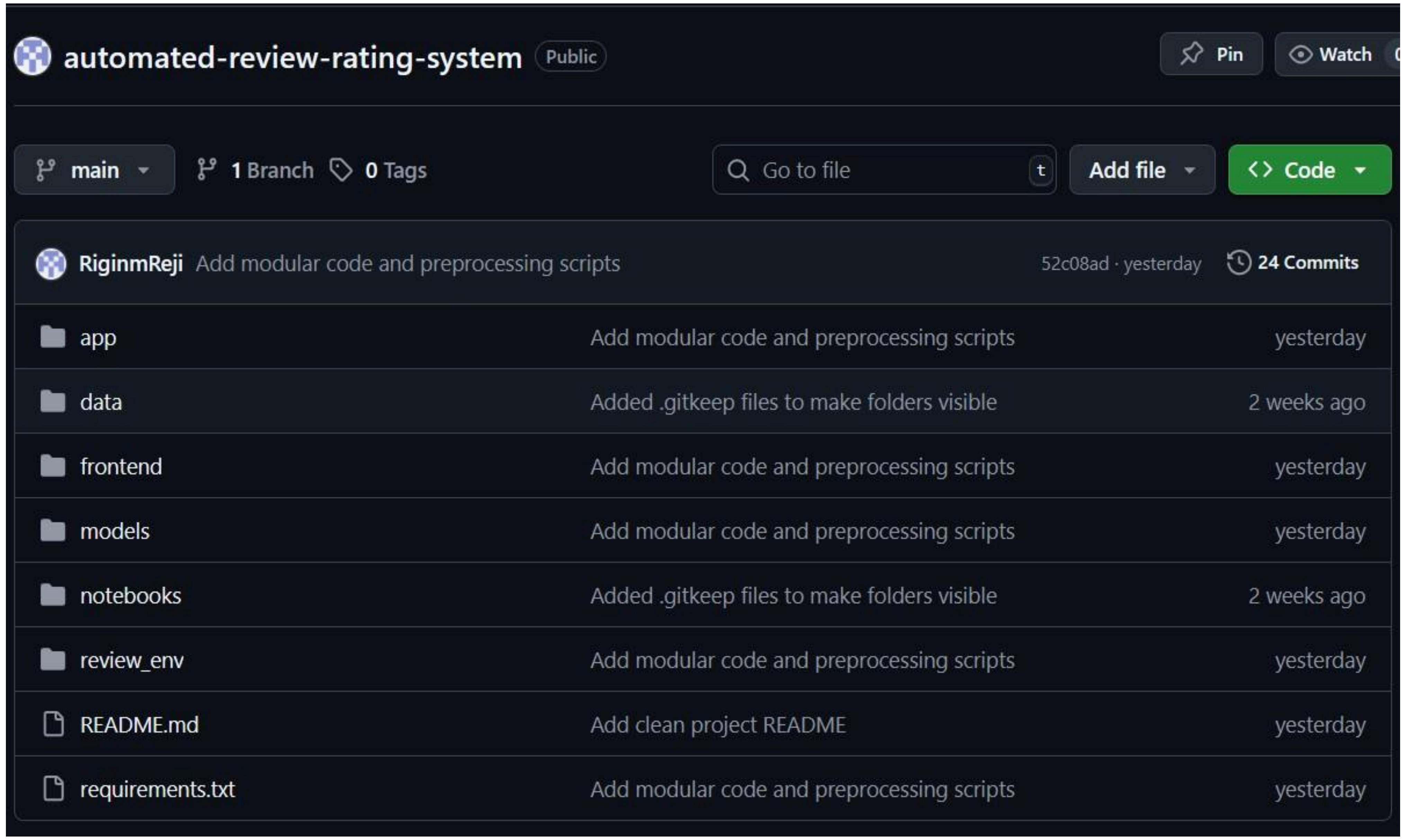
2. ENVIRONMENTAL SETUP

- Python: 3.9+
- Libraries: pandas, numpy, nltk, scikit-learn, re
- NLTK Resources: stopwords, punkt, wordnet
- IDE: Google Colab and VS Code
- Hardware: Laptop

3.GITHUB PROJECT SETUP

Created GitHub repository : **automated-review-rating-system**

Structure of directory



RiginmReji Add modular code and preprocessing scripts 52c08ad · yesterday 24 Commits

app	Add modular code and preprocessing scripts	yesterday
data	Added .gitkeep files to make folders visible	2 weeks ago
frontend	Add modular code and preprocessing scripts	yesterday
models	Add modular code and preprocessing scripts	yesterday
notebooks	Added .gitkeep files to make folders visible	2 weeks ago
review_env	Add modular code and preprocessing scripts	yesterday
README.md	Add clean project README	yesterday
requirements.txt	Add modular code and preprocessing scripts	yesterday

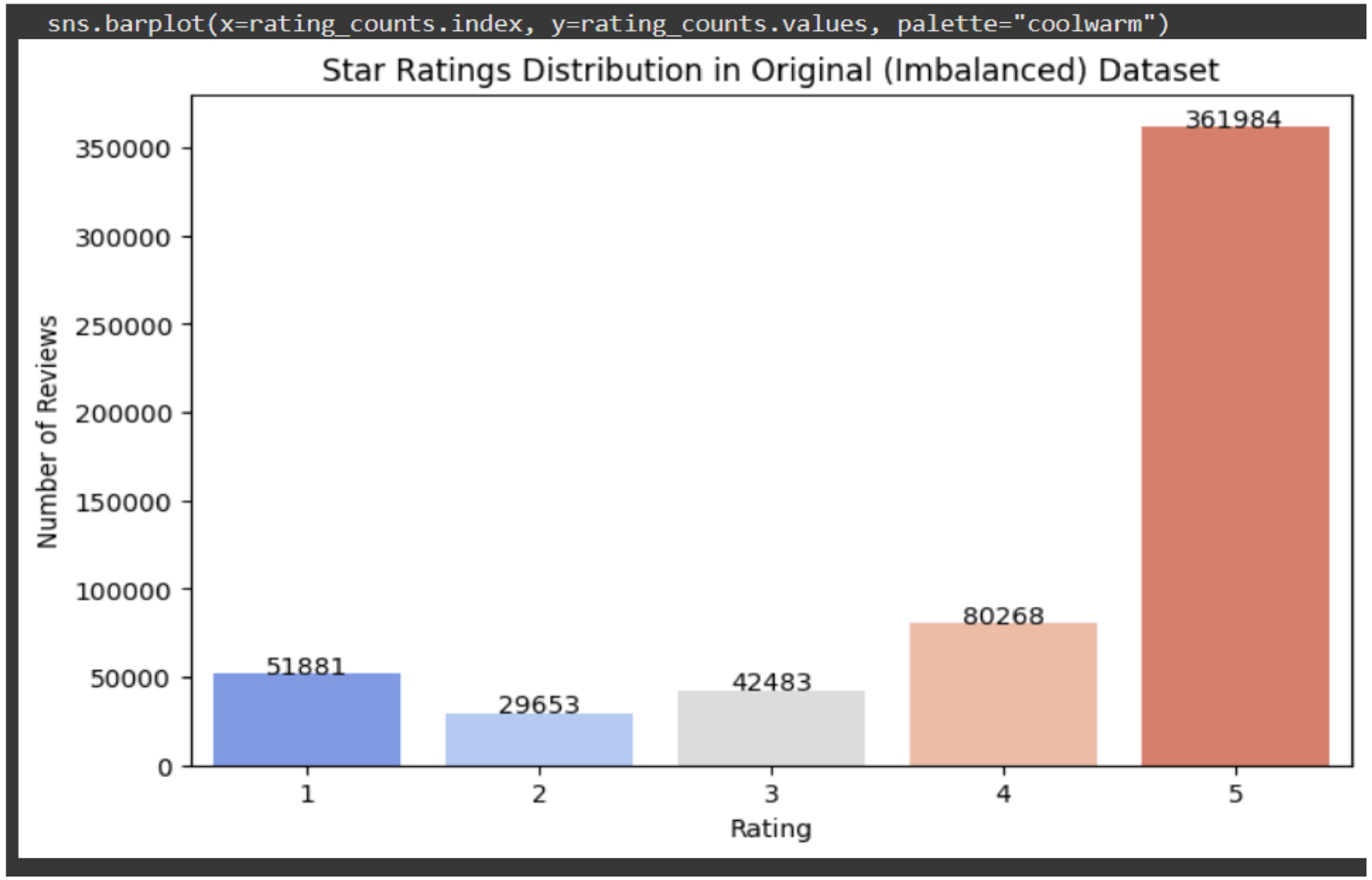
4.DATA COLLECTION

- The dataset was collected from Kaggle, containing real-world customer reviews from e-commerce platforms.
- It has 542454 rows and 14 columns
- Dataset link = https://www.kaggle.com/snap/amazon-fine-food-reviews/downloads/amazon-fine-food-reviews.zip
- It includes the review text and corresponding star ratings (1–5), along with optional metadata such as product IDs and timestamps
- Initial preprocessing involved removing duplicates, handling missing values, and filtering extremely short or long reviews to ensure quality for analysis and modeling.
- Final dataset contains 2 columns with Rating and Review_text
- 1 star -33,308, 2 star - 17,804, 3 star – 26,713, 4 star - 53,068, 5 star - 247801

4.2.2 Imbalanced dataset

Link for imbalanced dataset =

- Imbalanced dataset was created with 1 star – 10%, 2 star – 15%, 3 star – 15%, 4 star – 30% , 5 star – 20%



```
sns.barplot(x=rating_counts.index, y=rating_counts.values, palette="coolwarm")
```

5.DATA PREPROCESSING

Effective data preprocessing is essential to improve the performance and reliability of machine learning models, especially when dealing with imbalanced data. The following techniques were applied to prepare the remaining review dataset (not included in the balanced set) for modeling.

5.1. Removing duplicates

Duplicate rows containing the exact same review and rating were removed from the remaining data to prevent bias and overfitting. This ensured that the **imbalanced dataset** only includes unique observations not present in the balanced dataset.

Code:

```
before = imbalanced_df.shape[0]
imbalanced_df = imbalanced_df.drop_duplicates(subset=['Review_text'])
after = imbalanced_df.shape[0]

print(f"Removed {before - after} duplicate reviews from the imbalanced dataset.")
print(f"Remaining reviews: {after}")
```

5.2. Removing conflicting reviews

Some reviews in the remaining data had identical text but different star ratings. These inconsistencies could confuse the model. Such conflicting entries were identified and removed to maintain **label clarity** in the imbalanced dataset.

Code:

```
conflicts = imbalanced_df.groupby("Review_text")["Rating"].nunique()
conflicting_reviews = conflicts[conflicts > 1]

print("Number of conflicting reviews in the imbalanced dataset:", conflicting_reviews.shape[0])
```

5.3 Handling missing values

Rows with missing or null values, particularly in the review text or rating column, were removed from the remaining data. This step ensured the **imbalanced dataset** was complete and meaningful for analysis.

Code:

```
print(imbalanced_df.isnull().sum())


imbalanced_df = imbalanced_df.dropna(subset=["Review_text"])
imbalanced_df = imbalanced_df[imbalanced_df["Review_text"].str.strip() != ""]


print("Shape of the imbalanced dataset after handling missing values:", imbalanced_df.shape)
print("Shape of the imbalanced dataset after handling missing values:", imbalanced_df.shape)
```

5.4 Dropping unnecessary columns

Non-essential columns such as user IDs, product ID, summary, and HelpfulnessNumerator were dropped from the remaining data. These fields did not contribute to the model and could introduce noise or privacy concerns in the imbalanced dataset.

5.5 Lowercasing Text

All review text in the imbalanced dataset was converted to lowercase to maintain uniformity. This helps prevent duplication of tokens, ensuring that words like "Good" and "good" are treated as the same word during analysis.

5.6 Removing URLs

URLs present in the review text were removed using regular expressions.

5.7 Removing emojis and Special characters

Emojis and special symbols may not contribute meaningful context for text analysis (unless specifically required). We remove these characters to focus on the core textual content.

5.8 Removing punctuations

Emojis and special symbols may not contribute meaningful context for text analysis (unless specifically required). We remove these characters to focus on the core textual content.

5.9 Stopwords Removal

Stopwords are commonly used words (like the, is, and, to, in) that usually carry little meaningful information in text analysis. Removing stopwords helps reduce noise and improves the performance of text-based models.

Example:

Before stopword removal:

"This is a very good product and I like it a lot" After

stopword removal:

"This very good product like lot"

Code:

```
import nltk
from nltk.corpus import stopwords

nltk.download("stopwords")
```

```
stop_words = set(stopwords.words("english"))

# Remove stopwords from the imbalanced dataset
imbalanced_df["Review_text"] = imbalanced_df["Review_text"].apply(
    lambda x: " ".join([word for word in x.split() if word.lower() not in stop_words])
)

print("After stopwords removal in the imbalanced dataset:")
print(imbalanced_df["Review_text"].head(10))
```

5.10 Lemmatization

Lemmatization is the process of converting words in the **imbalanced dataset** into their base or dictionary form, known as a lemma. Unlike stemming (which simply chops word endings), lemmatization considers the context and part of speech, producing meaningful root words and improving text consistency for modeling.

Example::

- “running” → “run”
- “better” → “good”
- “cars” → “car”

why lemmatization is better than stemming ?

Meaningful Output:

Stemming just chops off word endings, which may produce non-words.

- “studies” → “studi”
- Lemmatization uses a dictionary to return valid base words.

“studies” → “study”

- Stemming = speed, Lemmatization = correctness.

5.11 Filtering by wordcount

Very short reviews might not contain enough context to be useful, and excessively long reviews could be outliers. We apply filtering to exclude:

- Reviews with fewer than 3 words
- Reviews that exceed 300 words

This ensure that the dataset remains robust and relevant for model training

6. Data visualisation

Box Plot

Box Plot is a visualization that shows the spread and distribution of review lengths across ratings in the dataset. It highlights the median review length, the range of most reviews, and detects outliers such as unusually short or long reviews. This helps in understanding how review size varies with customer ratings.

1.sample of 1 star rating

1.

“This is hands down the worst product I’ve ever purchased. It stopped working within a week, and the quality is absolutely terrible. It looks nothing like the pictures. Customer service was completely unhelpful, and they refused to issue a refund. Total waste of money and time. Avoid this at all costs!”

2.

“I wish I could give zero stars. The item arrived broken, and the packaging was torn. When I tried to contact the seller, they didn’t respond at all. The product doesn’t function, and it feels like a cheap knockoff.

I’m extremely disappointed and regret buying this. Don’t fall for the positive reviews.”

3.

“This product is a complete disaster. It doesn’t do what it claims, and it feels like it’s made from the cheapest materials possible. It stopped working after two uses. The company should be ashamed to sell something like this. Very frustrating experience — never buying from this brand again!”

2.sample of 2 star rating

1.
“I’m not very happy with this purchase. The product looked great online, but in reality, it feels cheap and poorly made. It stopped working properly after just a few days of use. Customer service didn’t respond to my complaint. For the price I paid, I expected better quality. Wouldn’t recommend it.”

2.
“The item arrived late, and the packaging was slightly damaged. It works, but only partially. Some features don’t function as promised. The material feels fragile and doesn’t look durable at all. It’s disappointing because I had high expectations based on the reviews. Not worth the money.”

3.
“The product quality is below average. It doesn’t perform consistently and often malfunctions. I tried contacting support but got no helpful response. It’s frustrating to spend money on something that doesn’t deliver even basic performance. I’ll probably look for another brand next time.”

3.sample of 3 star rating

1.
“The product is okay, but it didn’t really impress me. It works as described, but the quality feels just average. It’s functional, but you can find similar options for the same price. Packaging was fine, and delivery was on time, but overall it just didn’t stand out. Not bad, not great — just okay.”

2.
“I had high hopes for this, but it’s pretty standard. It does the job, but nothing about it feels premium or special. The design looks decent, and it performs adequately. It’s fine for short-term use but may not last long. It’s worth buying only if you’re on a tight budget.”

3.
“It’s an average product with both pros and cons. Works fine, but the performance is inconsistent at times. I had to restart it a few times to get it to function properly. The price is fair, but I expected a bit more durability. Not disappointed, but not excited either.”

4.sample of 4 star rating

1.
“I’m quite satisfied with this product. It does what it promises and looks great too. The performance is good, though I did notice a slight lag at times. Nothing major, just a small flaw. Packaging was decent and delivery was quick. Overall, a solid product that I would recommend, just wish it had a few more features for the price.”
2.
“Very good quality and design. It works well for my needs, but there were some minor issues with setup. Once I figured it out, it’s been smooth sailing. The materials feel durable and the build quality is nice. For the price point, it’s definitely a worthwhile purchase. Not perfect, but very close!”
3.
“I’ve been using this for about a month now, and I’m quite happy with it. The performance is consistent, and it’s easy to operate. One small issue I faced was with the instruction manual — it wasn’t very clear. Aside from that, no complaints. It’s a great product and well worth the money!”

5.sample of 5 star rating

1.
“I’m absolutely thrilled with this product! From the moment it arrived, I could tell it was made with quality materials. The packaging was neat, and the product looked even better than in the pictures. I’ve been using it daily for two weeks now, and it performs flawlessly. It’s user-friendly, efficient, and definitely worth every penny. I highly recommend this to anyone looking for reliability and great value!”

2.
“This is by far one of the best purchases I’ve made online. The product works exactly as described, maybe even better! It’s sturdy, stylish, and super easy to use. The customer service was also excellent — they responded to my questions quickly and were very polite. Shipping was fast too. Overall, an amazing experience. I’ll surely buy again from this brand!”

3.
“I rarely write reviews, but this product deserves one! The performance is outstanding, and it has made my daily routine so much easier. The attention to detail in design and functionality is impressive. It feels premium, works efficiently, and the price is very reasonable for the quality you get. Truly a 5-star experience!”

7. Train-Test-Split

 Training Set (typically 80%): Used to train the machine learning model.
 Test Set (typically 20%): Used to evaluate the model’s performance on unseen data.

 Even with an **imbalanced dataset**, splitting ensures that both sets follow the same **imbalanced class distribution**.

Why stratified split?

 In imbalanced classification problems, some classes (like **1-star or 2-star reviews**) are much smaller than others (like **5-star reviews**).

 If we split randomly **without stratification**, the smaller classes might be missing or underrepresented in the test set.

 Stratified splitting makes sure the **same imbalance ratio** is preserved in both training and test sets.

 This gives a **realistic evaluation**, since the model is tested under the same imbalance conditions as it was trained on.

How it was done?

To prepare the data for model training, the dataset was first shuffled randomly to eliminate any order bias. A **stratified train-test split** was then performed using train_test_split() from sklearn.model_selection with the stratify=y argument to ensure that the **intentionally imbalanced distribution of star ratings** was preserved in both training and testing sets. The dataset was divided into an **80% training set and 20% testing set**. After splitting, all text preprocessing steps—**lowercasing, stopword removal, lemmatization, and cleaning**—were applied **separately** to X_train and X_test to avoid data leakage and ensure model integrity

code:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

def prepare_train_test(df, text_col="Review_text", target_col="Rating",
                       test_size=0.2, max_features=5000):
    """
    Perform stratified train-test split and TF-IDF vectorization
    while preserving imbalanced class distribution.

    Args:
        df (pd.DataFrame): Input dataframe (can be imbalanced).
        text_col (str): Column with text data.
        target_col (str): Target label column.
        test_size (float): Proportion of test split.
        max_features (int): Number of TF-IDF features.

    Returns:
```

```
    X_train_vec, X_test_vec, y_train, y_test, vectorizer
"""

# Train-test split (stratified ensures imbalanced proportions are preserved in both sets)
X = df[text_col]
y = df[target_col]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=test_size, stratify=y, random_state=42


)
```

<u>Tokenization & Embedding)</u>

For the BiLSTM model, the review text was first **tokenized**, converting each review into a sequence of word indices. This allows the model to understand the order of words in the text.

The sequences were then passed through an **embedding layer**, which maps each word index to a dense vector representation. This helps the model capture semantic meaning and relationships between words, providing a rich input for the BiLSTM to learn from.

Code:

```
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

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')
```