# AUTOMATED REVIEW RATING SYSTEM

## INTRODUCTION

The Automated Review Rating System is a Machine Learning–based project that predicts a product/service rating based on a customer's written review. Instead of manually rating after writing feedback, the system analyzes the text and automatically generates a rating score.

With the rapid expansion of e-commerce and digital service platforms, user-generated reviews have become a vital source of feedback for both businesses and consumers. These reviews contain valuable information about customer satisfaction, product quality, and service experience. However, the sheer volume of textual feedback makes manual evaluation time-consuming, subjective, and inefficient. Additionally, inconsistencies between written reviews and their corresponding star ratings can mislead both customers and organizations.

The Automated Review Rating System aims to bridge this gap by predicting the appropriate rating (1–5 stars) from a customer's review text using Natural Language Processing (NLP) and Machine Learning. Instead of relying solely on user-given ratings, the system reads the review, interprets sentiment and context, and automatically generates the most suitable rating.

Through this automation, the project enables:

- Faster and more accurate sentiment evaluation
- Reduced dependency on subjective or fake ratings
- Large-scale analysis of customer opinions

## ENVIORMENT SETUP

To create a stable development environment for the project:

**Installed Python (Anaconda Recommended)**

- Downloaded and installed Anaconda from the official site-(https://www.anaconda.com)

**Set up Jupyter Notebook**

- Opened **Anaconda Navigator → Launched Jupyter Notebook**
- Verified Jupyter Interface is working

**Libraries Required**

The *Automated Review Rating System* relies on a set of Python libraries for data handling, visualization, Natural Language Processing, and Machine Learning. Each library plays a specific role in processing text data and building the rating prediction model.

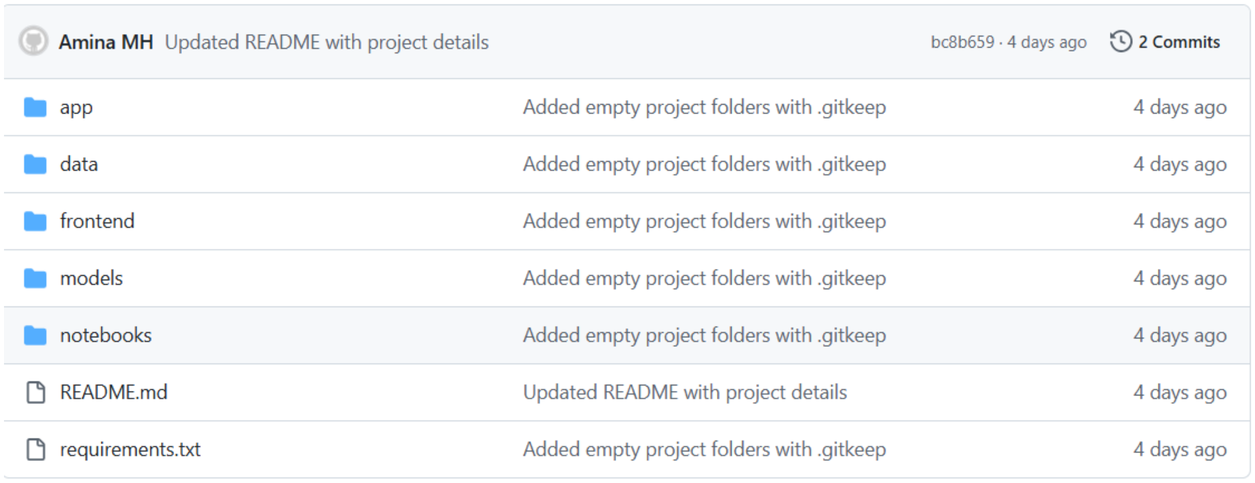
- pandas
- numpy
- matplotlib
- seaborn
- sickit-learn
- nltk(Natural Language Toolkit)

**GitHub Repository Setup**

To enable version control and collaborative development for the *Automated Review Rating System*, a GitHub repository was created and configured with the required project structure.

Repository Name: https://github.com/aminamh-29/automated-review-rating-system.git

**Folder Structure**



Amina MH Updated README with project details bc8b659 · 4 days ago 2 Commits

| | | |
|---|---|---|
| app | Added empty project folders with .gitkeep | 4 days ago |
| data | Added empty project folders with .gitkeep | 4 days ago |
| frontend | Added empty project folders with .gitkeep | 4 days ago |
| models | Added empty project folders with .gitkeep | 4 days ago |
| notebooks | Added empty project folders with .gitkeep | 4 days ago |
| README.md | Updated README with project details | 4 days ago |
| requirements.txt | Added empty project folders with .gitkeep | 4 days ago |

- Added a based folder structure.
- Create an initial README.md with short description of the project.
- Included requirements.txt for setting up the environment.

## 1. DATA COLLECTION

The goal of this task is to gather real-world review datasets to build and train the Automated Review Rating System

**Dataset Overview:**

- **Source-**kaggle Amazon reviews
- **Load dataset:** Read CSV into a Pandas DataFrame.
- **Dataset overview:** Check rows, columns, data types, and summary statistics.

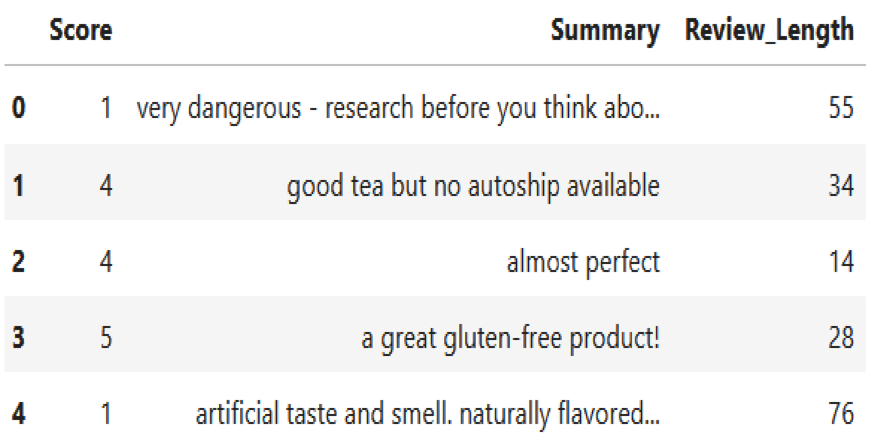
- **Drop unnecessary columns:** Keep only the relevant columns (`summary` and `score`) for modeling.
- **Remove duplicates:** Ensure each review is unique to avoid bias.
- **Check for null values:** Identify missing data and calculate percentages.
- **Handle nulls:**
- **Dataset link-**
- **Balanced dataset link-**
- **Imbalanced dataset link:-**

## 2. DATASET PRE-PROCESSING

Text preprocessing is a critical step in preparing the dataset for Machine Learning. Raw review data often contains noise, inconsistencies, and irrelevant information that can negatively impact model performance. The preprocessing pipeline ensures that reviews are clean, standardized, and machine-readable.

### Lower Case

```
df["Summary"] = df["Summary"].astype(str).str.lower()

 df.head()
```

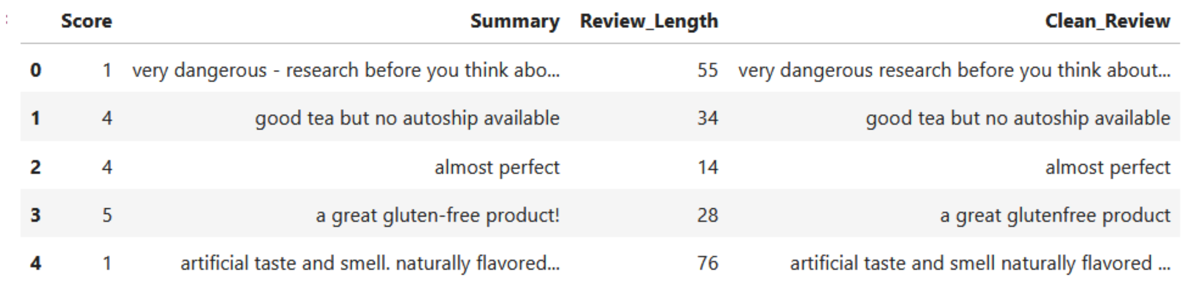


|   | Score | Summary | Review_Length |
|---|---|---|---|
| 0 | 1 | very dangerous - research before you think abo... | 55 |
| 1 | 4 | good tea but no autoship available | 34 |
| 2 | 4 | almost perfect | 14 |
| 3 | 5 | a great gluten-free product! | 28 |
| 4 | 1 | artificial taste and smell. naturally flavored... | 76 |

### Remove URLs, HTML tags, emojis, punctuation and special characters

```
import re

  def clean_text(text):

      text = str(text)

      text = re.sub(r"http\S+|www\S+|https\S+", "", text)          # remove URLs
```

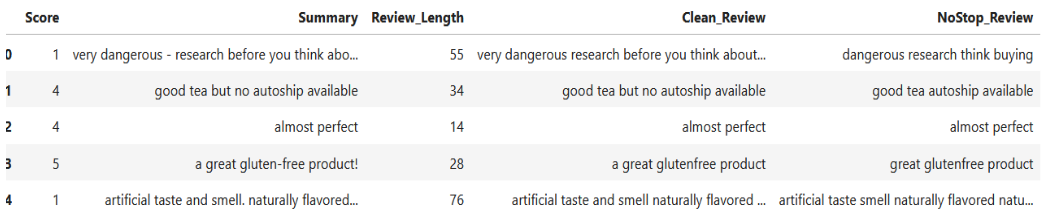
```
    text = re.sub(r"<.*?>", "", text)                    # remove HTML tags

    text = re.sub(r"[^\w\s]", "", text)                  # remove punctuation & special
characters

    text = re.sub(r"[\U0001F600-\U0001F64F"

                r"\U0001F300-\U0001F5FF"

                r"\U0001F680-\U0001F6FF"

                r"\U0001F1E0-\U0001F1FF]+", "", text)        # remove emojis

    text = re.sub(r"\s+", " ", text).strip()             # remove extra spaces

    return text

df["Clean_Review"] = df["Summary"].apply(clean_text)

df.head()
```



| | Score | Summary | Review_Length | Clean_Review |
|---|---|---|---|---|
| 0 | 1 | very dangerous - research before you think abo... | 55 | very dangerous research before you think about... |
| 1 | 4 | good tea but no autoship available | 34 | good tea but no autoship available |
| 2 | 4 | almost perfect | 14 | almost perfect |
| 3 | 5 | a great gluten-free product! | 28 | a great glutenfree product |
| 4 | 1 | artificial taste and smell. naturally flavored... | 76 | artificial taste and smell naturally flavored ... |

**Remove Stopwords**

```
import nltk

import sys

import io

from contextlib import redirect_stdout

# Suppress NLTK download messages

f = io.StringIO()

with redirect_stdout(f):
    try:
        nltk.data.find('corpora/stopwords')
    except LookupError:
        nltk.download('stopwords', quiet=True)
from nltk.corpus import stopwords
#remove stopwords
stop_words = set(stopwords.words("english"))


def remove_stopwords(text):
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    return " ".join(filtered_words)
df["NoStop_Review"] = df["Clean_Review"].apply(remove_stopwords)


df.head()
```

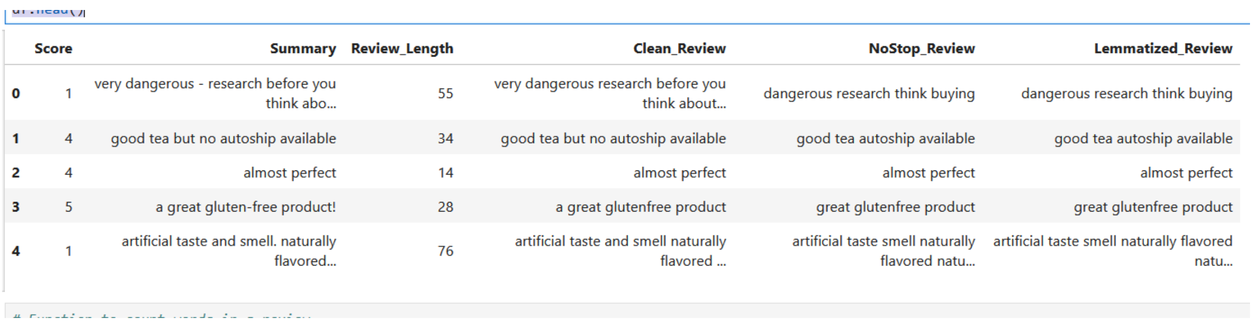


| | Score | Summary | Review_Length | Clean_Review | NoStop_Review |
|---|---|---|---|---|---|
| 0 | 1 | very dangerous - research before you think abo... | 55 | very dangerous research before you think about... | dangerous research think buying |
| 1 | 4 | good tea but no autoship available | 34 | good tea but no autoship available | good tea autoship available |
| 2 | 4 | almost perfect | 14 | almost perfect | almost perfect |
| 3 | 5 | a great gluten-free product! | 28 | a great glutenfree product | great glutenfree product |
| 4 | 1 | artificial taste and smell. naturally flavored... | 76 | artificial taste and smell naturally flavored ... | artificial taste smell naturally flavored natu... |

**Lemmatization**

```
import nltk
nltk.download("wordnet", quiet=True)
nltk.download("omw-1.4", quiet=True)
nltk.download("punkt", quiet=True)
```

```
from nltk.stem import WordNetLemmatizer

from nltk.tokenize import word_tokenize


lemmatizer = WordNetLemmatizer()


def apply_lemmatization(text):

    words = word_tokenize(text)

    lemmatized_words = [lemmatizer.lemmatize(w) for w in words]

    return " ".join(lemmatized_words)


df["Lemmatized_Review"] = df["NoStop_Review"].apply(apply_lemmatization)

df.head()
```



| | Score | Summary | Review_Length | Clean_Review | NoStop_Review | Lemmatized_Review |
|---|---|---|---|---|---|---|
| 0 | 1 | very dangerous - research before you think abo... | 55 | very dangerous research before you think about... | dangerous research think buying | dangerous research think buying |
| 1 | 4 | good tea but no autoship available | 34 | good tea but no autoship available | good tea autoship available | good tea autoship available |
| 2 | 4 | almost perfect | 14 | almost perfect | almost perfect | almost perfect |
| 3 | 5 | a great gluten-free product! | 28 | a great glutenfree product | great glutenfree product | great glutenfree product |
| 4 | 1 | artificial taste and smell. naturally flavored... | 76 | artificial taste and smell naturally flavored ... | artificial taste smell naturally flavored natu... | artificial taste smell naturally flavored natu... |

**Why Lemmatization Instead of Stemming**

Both **stemming** and **lemmatization** are techniques to reduce words to their base form, but they differ in **accuracy and meaning:**

Reason for Choosing Lemmatization

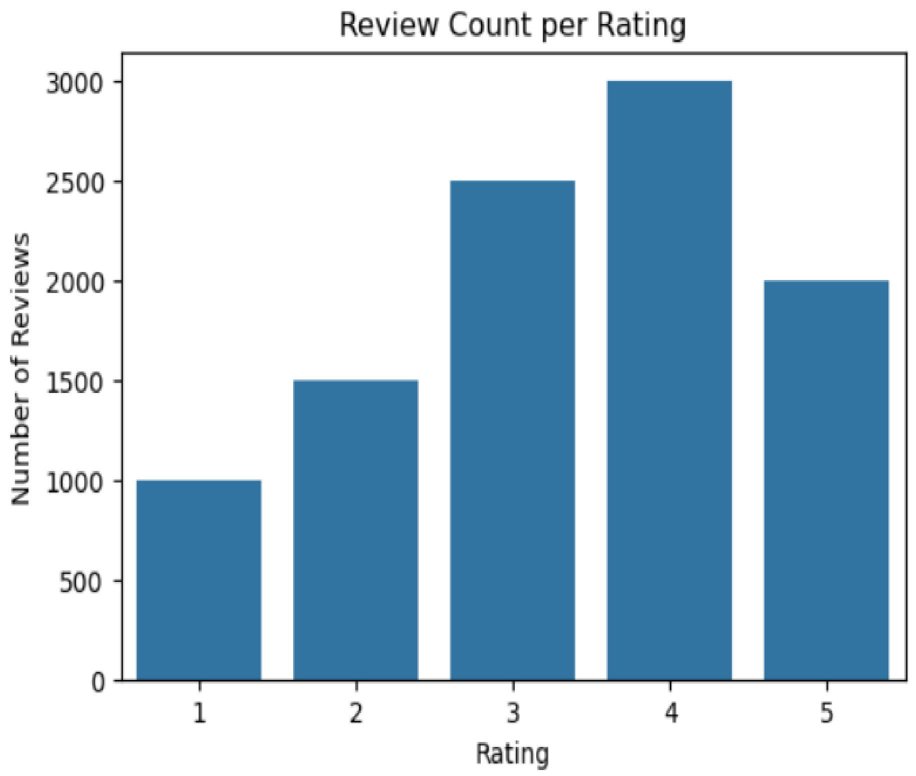
1. **Preserves semantic meaning:** Words remain meaningful for human interpretation and machine learning.
2. **Reduces noise in text:** Unlike stemming, lemmatization avoids confusing variants like stud, run, go that may not match the original meaning.
3. **Better for sentiment analysis:** Helps the model understand that different forms of a word *(*good, better, best*)* are related, improving accuracy in predicting review ratings.

## 3. UNBALANCED DATASET

The dataset contains customer review texts along with their corresponding star ratings (1–5). During exploratory data analysis, it was observed that the distribution of review ratings is not uniform across all classes, leading to an unbalanced dataset.

### Class Imbalance Findings

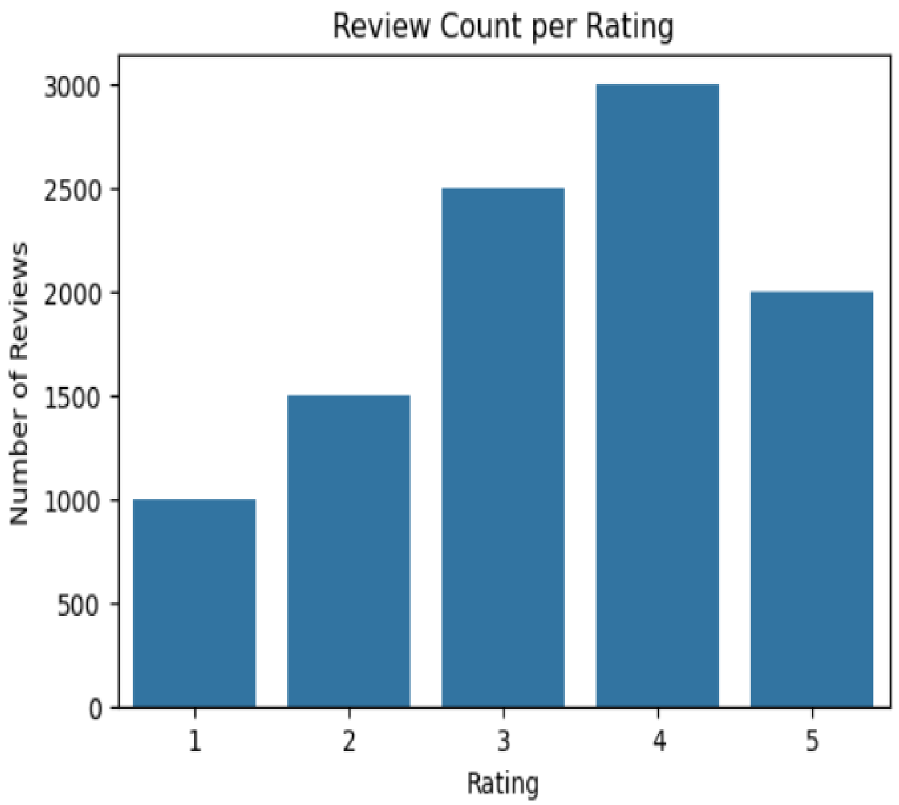
- Certain ratings appear significantly more frequently than others.
- Typically, higher ratings (e.g., 4 and 5 stars) are dominant in comparison to lower ratings (e.g., 1 and 2 stars).
- This imbalance reflects real-world behavior, where customers are more likely to leave positive reviews rather than negative ones.
- Review count Bar Plot



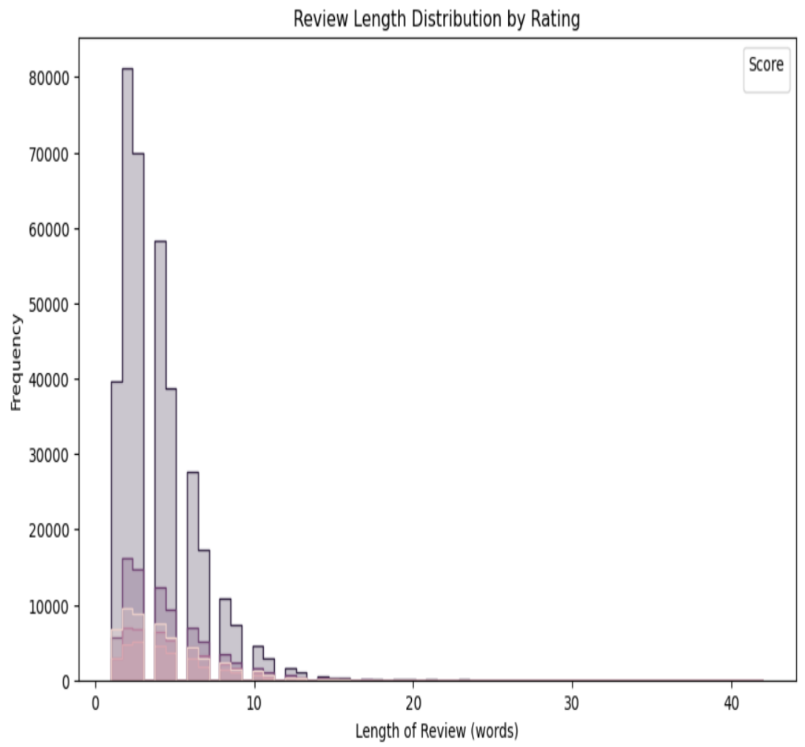
## 4. DATA VISUALIZATION

Data visualization was performed to gain meaningful insights from the Amazon reviews dataset before and after preprocessing. The visual analysis helped identify trends, detect noise/outliers, and make informed decisions for data balancing and text cleaning

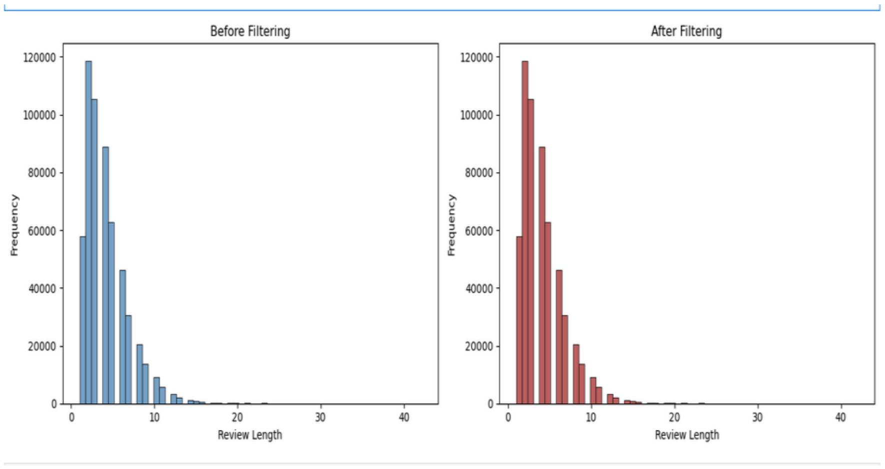
- REVIEW COUNT PER RATING



- REVIEW LENGTH



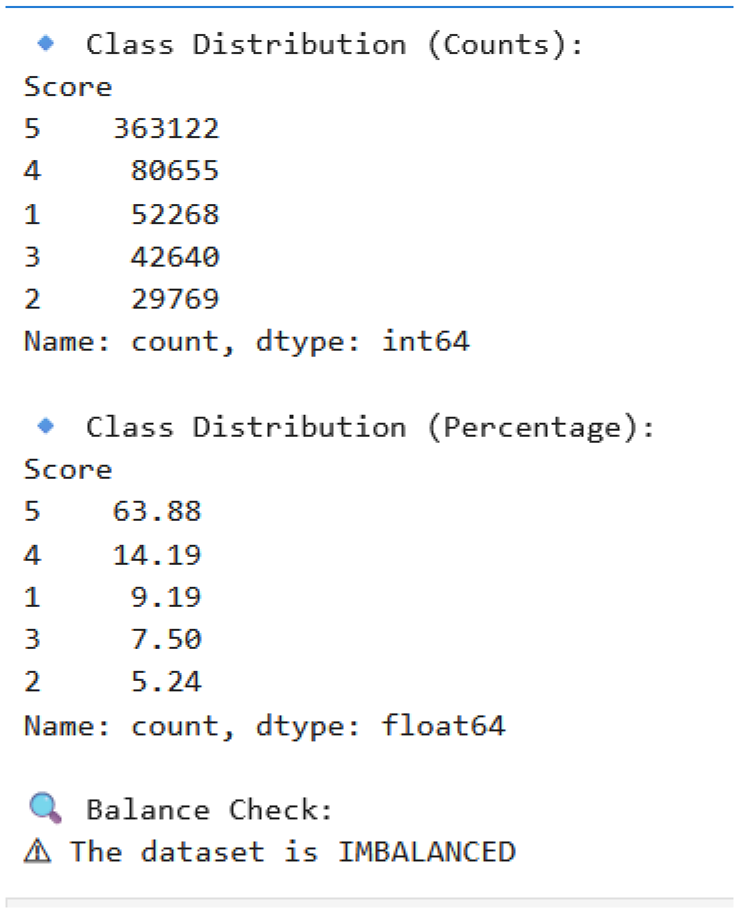
- BEFORE AND AFTER FILTERING



- IMBALANCE DATASET

The dataset used for this project exhibited a significant class imbalance across review ratings. A majority of the reviews belonged to 4- and 5-star categories**,** while 1- and 2-star reviews were highly under-represented. This imbalance can bias the prediction model toward majority classes, reducing its ability to correctly identify low-rating reviews.

```
# Class distribution counts

class_counts = df['Score'].value_counts()



print("🔹 Class Distribution (Counts):")

print(class_counts)

print("\n🔹 Class Distribution (Percentage):")

print((class_counts / len(df) * 100).round(2))



# Automatic balance check

ratio = class_counts.max() / class_counts.min()



print("\n🔍 Balance Check:")

if ratio > 1.5:
    print("⚠ The dataset is IMBALANCED")
else:
    print("✓ The dataset is BALANCED")
```



```
 🔹 Class Distribution (Counts):
Score
5    363122
4     80655
1     52268
3     42640
2     29769
Name: count, dtype: int64

 🔹 Class Distribution (Percentage):
Score
5    63.88
4    14.19
1     9.19
3     7.50
2     5.24
Name: count, dtype: float64

 🔍 Balance Check:
⚠ The dataset is IMBALANCED
```

5. TRAIN TEST SPLIT

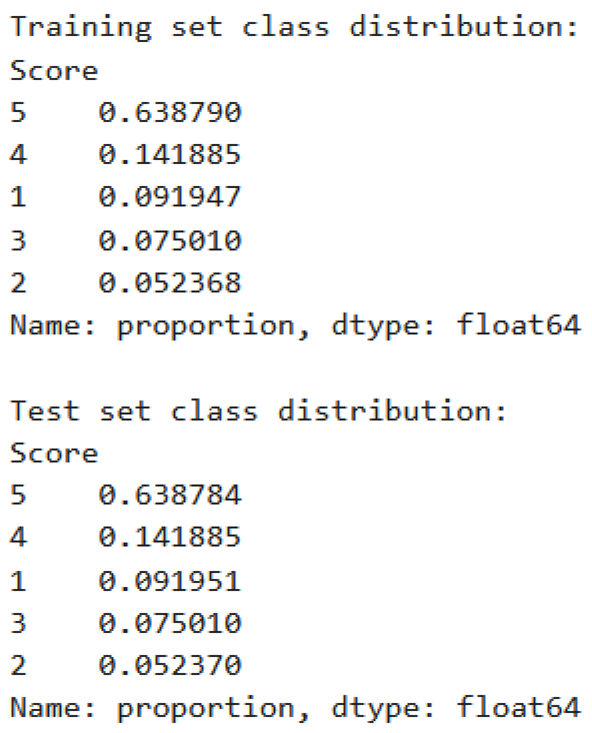
To evaluate the model's ability to generalize to unseen reviews, the dataset was divide into training and testing subsets**.** This ensures fair performance measurement and prevents information leakage. Train–Test Split is a method used in machine learning to divide a dataset into two separate parts**:**

**Training Set** — the portion of data used to teach the model
**Testing Set** — the portion of data used to evaluate the model

**STRATIFIED SPLITTING**

Stratified splitting is a technique used during the train–test split to ensure that the proportion of each class in the target variable remains consistent in both the training and testing datasets. Instead of randomly dividing the dataset, stratification preserves the original class distribution**.** In the context of this project, the target variable is the **review** rating (1–5 stars). Since the dataset is imbalanced, some ratings appear far more frequently than others—especially 5-star reviews.

```
from sklearn.model_selection import train_test_split


# X = text column (features), y = rating/score column (labels)

X = df["Summary"]     # or Clean_Review / Lemma_Review if already preprocessed

y = df["Score"]


# Stratified Split — ensures balance across rating classes

X_train, X_test, y_train, y_test = train_test_split(

    X, y,

    test_size=0.20,

    stratify=y,

    shuffle=True,

    random_state=42

)


# Final output

print("Final Output for Train–Test Split:")

print("X_train size:", len(X_train))

print("X_test size:", len(X_test))

print("y_train size:", len(y_train))

print("y_test size:", len(y_test))
```



```
Training set class distribution:
Score
5    0.638790
4    0.141885
1    0.091947
3    0.075010
2    0.052368
Name: proportion, dtype: float64

Test set class distribution:
Score
5    0.638784
4    0.141885
1    0.091951
3    0.075010
2    0.052370
Name: proportion, dtype: float64
```

## 6. VECTORIZATION

Vectorization is the process of converting text data (words, sentences, reviews*)* into **numeric features** that a model can analyze and learn from. In this project, customer review text is transformed into a numerical representation before training the rating prediction model. Machine learning models cannot understand raw text directly — they require numerical input**.**

Why Vectorization is Needed
Text data: Contains vocabulary, grammar, and context.
Cannot be processed by machine learning algorithms in its raw form
Vectorization: Converts text into numbers
Preserves word importance and meaning patterns
Enables algorithms to detect sentiment and rating-related signals

```
#5:vectorization

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer



# ---------------------------------------------------------------------
# SECTION 1 — PREPROCESSING FUNCTIONS

# --------------------------------------------------------------------

def clean_text(text):

    """Lowercase, remove punctuation, numbers, and extra spaces."""

    text = str(text).lower()

    text = re.sub(r'[^a-z\s]', ' ', text)

    text = re.sub(r'\s+', ' ', text).strip()

    return text

from sklearn.feature_extraction.text import TfidfVectorizer



# Initialize TF-IDF vectorizer

vectorizer = TfidfVectorizer(max_features=5000)   # you can adjust max_features if
needed



# Fit only on TRAIN data

X_train_vec = vectorizer.fit_transform(X_train)



# Transform TEST data using the same vocabulary learned from train

X_test_vec = vectorizer.transform(X_test)



print("Vectorization completed successfully!")

print("Training matrix shape:", X_train_vec.shape)

print("Testing matrix shape:", X_test_vec.shape)
```

7. **TF-IDF**

TF-IDF is a text vectorization technique used to convert words into numerical values based on how important or informative they are in a document.

Equation

$$\text{TF-IDF}(t,d)=\text{TF}(t,d)\times\text{IDF}(t)$$

It measures two things:

- **Term Frequency (TF):** how often a word appears in a document
- **Inverse Document Frequency (IDF):** how rare or unique the word is across all documents

By combining both, TF-IDF gives higher weight to important words and lower weight to common, less meaningful words.

- **Why TF-IDF Instead of Bag of Words**

  TF-IDF is chosen over Bag of Words because it provides a **more informative representation of text** for machine learning models. While Bag of Words simply counts how often each word appears in a document, TF-IDF weighs words by their importance, giving higher scores to words that are rare and meaningful across the dataset, and lower scores to very common words that carry little unique information**.**

## 8. SAMPLE REVIEWS

```
============================================================
 Rating: 1 ★ | 10 Sample Reviews
============================================================
1. Sweet & Low without the cancer.
2. wedding mom
3. Don't waste your money or your Keurig on this!
4. MADE IN CHINA!!!
5. Tastes like cheap meat and salt
6. Buyer Beware of Flavor Changes Not Accurately Described
7. Don't Buy It!
8. Poor Packaging
9. Low. Low. I repeat: Low.
10. Company sold out. No different from regular grocery dog food.

============================================================
 Rating: 2 ★ | 10 Sample Reviews
============================================================
1. Eehhh...not that great but not terrible
2. Dry and hard
3. Cloyingly sweet
4. real candy big pieces?
5. Disappointing Taste and Texture
6. Fairly strange taste
7. Sub-par
```

```
8. I think the box I got was a dud
9. Everlasting Treat Does Not Last!
10. Poor Customer Service

============================================================
 Rating: 3 ⭐ | 10 Sample Reviews
============================================================
1. After taste
2. Bit Too Dry
3. A humorous yet creepy novelty soda...
4. Expect not to have crumbs
5. It's ok, but it's not a margarita, or even an alcohol-free margarita
(updated)
6. Ginger Lover says this beverage is OK
7. Sketchy on the flavor assortment
8. Taste kind of strange....
9. Not bad, but not good either
10. Nice taste, but too tough to chew for me.

============================================================
 Rating: 4 ⭐ | 10 Sample Reviews
============================================================
1. Crunchy and delicious!
2. pretty good
3. Bad Packaging
4. Bear Naked Granola...normally great, but...
5. Good product for a good price and my cat likes it.
6. Not SUPER-delish, but not that bad either.
7. flaxseed
8. Butler's chocolate is the real deal direct from Ireland
9. 3 small venus flytraps 3 inch pot
10. Prett good coconut milk

============================================================
 Rating: 5 ⭐ | 10 Sample Reviews
============================================================
1. The only hor chocolate that actually has milk!
2. Great buy! Great taste!
3. $7.09 - you got to be kidding
4. Great taste and texture
5. Soooooooooo good!
6. Magic in a Bottle
7. A tasty Pumpernickle
8. Awesome healthIER snack option!
9. My son loves it!
10. Kids favorite!
```

## 9.. SUMMARY

The Automated Review Rating System is designed to predict customer review ratings (1–5 stars) from textual review data. The system processes raw reviews through cleaning, tokenization, lemmatization, and vectorization (TF-IDF or Bag of Words) to convert text into meaningful numerical features suitable for machine learning.

The dataset was analyzed for class imbalance and adjusted to create a balanced dataset, ensuring that each rating class (1–5 stars) has roughly equal representation. This prevents the model from being biased toward majority classes and allows it to learn patterns from all ratings fairly.

Following balancing, the data was split into stratified training and test sets to maintain class distribution. Key decisions, such as choosing lemmatization over stemming and TF-IDF over Bag of Words**,** were made to improve semantic understanding and model performance.

By applying this pipeline, the system ensures:

- Effective feature extraction from text
- Fair and unbiased model training and evaluation