**Automated-review-rating-system**

The Automated Review Rating System is a Python-based application designed to analyse user-generated reviews and predict corresponding ratings using natural language processing and machine learning techniques. It streamlines the feedback analysis process for platforms such as e-commerce websites, hospitality services, or educational portals.

**Example Use Case**

Imagine a hotel receives hundreds of guest reviews daily. Manually reading and scoring each one is time-consuming and inconsistent. With this system:

- Textual reviews like “The rooms were spotless and staff was friendly” are automatically interpreted using sentiment analysis.

- The system predicts a 4.5-star rating based on language tone and keyword intensity.

- Review summaries and rating predictions are visualized for the management team to improve service.

This helps businesses uncover hidden insights, maintain consistent feedback metrics, and improve customer experience based on real-time review data.

**Project Overview and Objective**

This project aims to develop an automated system that predicts product review ratings (1 to 5 stars) based on the text of the review. Using machine learning techniques, the system analyses textual review data to learn patterns and accurately estimate the star rating.

The objective is to build a clean, balanced dataset, apply appropriate pre-processing, and train baseline models using text vectorization methods for initial prototyping.

**Dataset Description**

The dataset consists of customer product reviews collected from CSV file. Each review contains a text field (Text) and an associated rating (1 to 5 stars). The data varies in quality with noise such as URLs, HTML tags, emojis, and variable review lengths.

**Pre-processing Steps**

Text Cleaning: Convert reviews to lowercase; remove URLs, HTML tags, punctuation, emojis, and special characters.

**Initial Cleaning**

Remove missing ratings or reviews

df = df.dropna(subset=['Text', 'Score'])

**Lowercase, strip punctuation, remove short reviews**

df['Text'] = df['Text'].str.lower().str.replace('[^a-z ]', '', regex=True)

df = df[df['Text'].str.len() > 10]

df = df.drop\_duplicates()

**Remove duplicates**

df = df.drop\_duplicates()

**Clean review text: lowercase, remove non-letters**

df['Text'] = df['Text'].str.lower().str.replace('[^a-z ]', '', regex=True)

**Remove reviews shorter than 10 characters**

df = df[df['Text'].str.len() > 10]

**Convert ratings to integer, if they aren't already**

df['Score'] = df['Score'].astype(int)

**Save cleaned data**

df.to\_csv('data/cleaned\_dataset/cleaned\_data.csv', index=False)

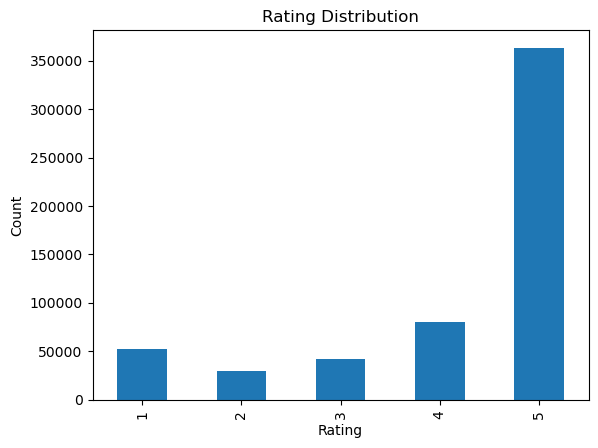
**Data Analysis & Visualization**

df['Score'].value\_counts().sort\_index().plot(kind='bar', title='Rating Distribution')

plt.xlabel('Rating')

plt.ylabel('Count')

plt.show()



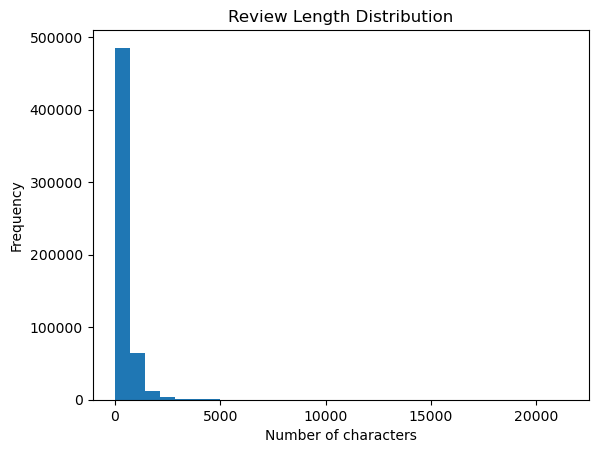
**Review Length Distribution**

df['review\_length'] = df['Text'].str.len()

df['review\_length'].plot(kind='hist', bins=30, title='Review Length Distribution')

plt.xlabel('Number of characters')

plt.show()

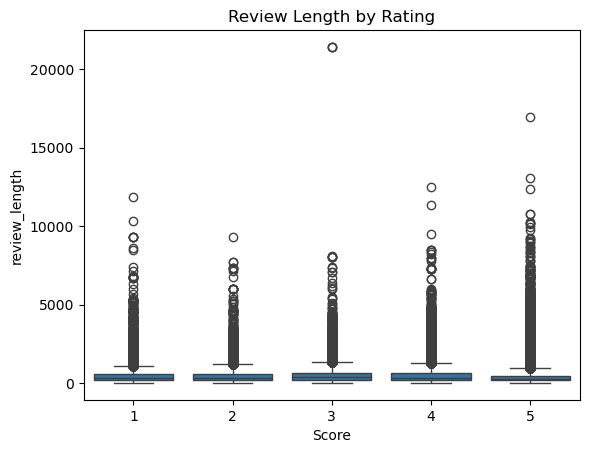


**Review Length by Rating**

sns.boxplot(x='Score', y='review\_length', data=df)

plt.title('Review Length by Rating')

plt.show()



**Violin plots**

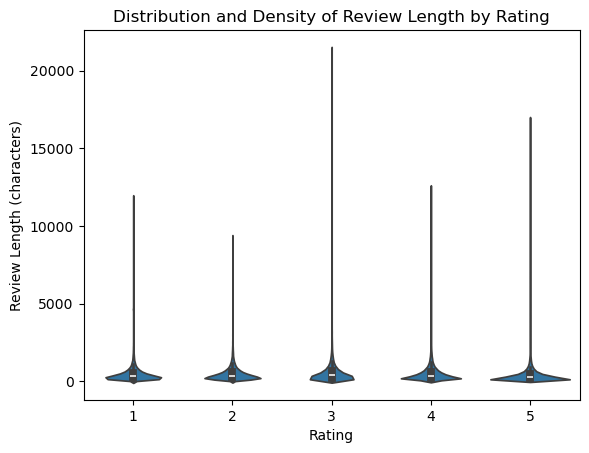
sns.violinplot(x='Score', y='review\_length', data=df, inner='box')

plt.title('Distribution and Density of Review Length by Rating')

plt.xlabel('Rating')

plt.ylabel('Review Length (characters)')

plt.show()



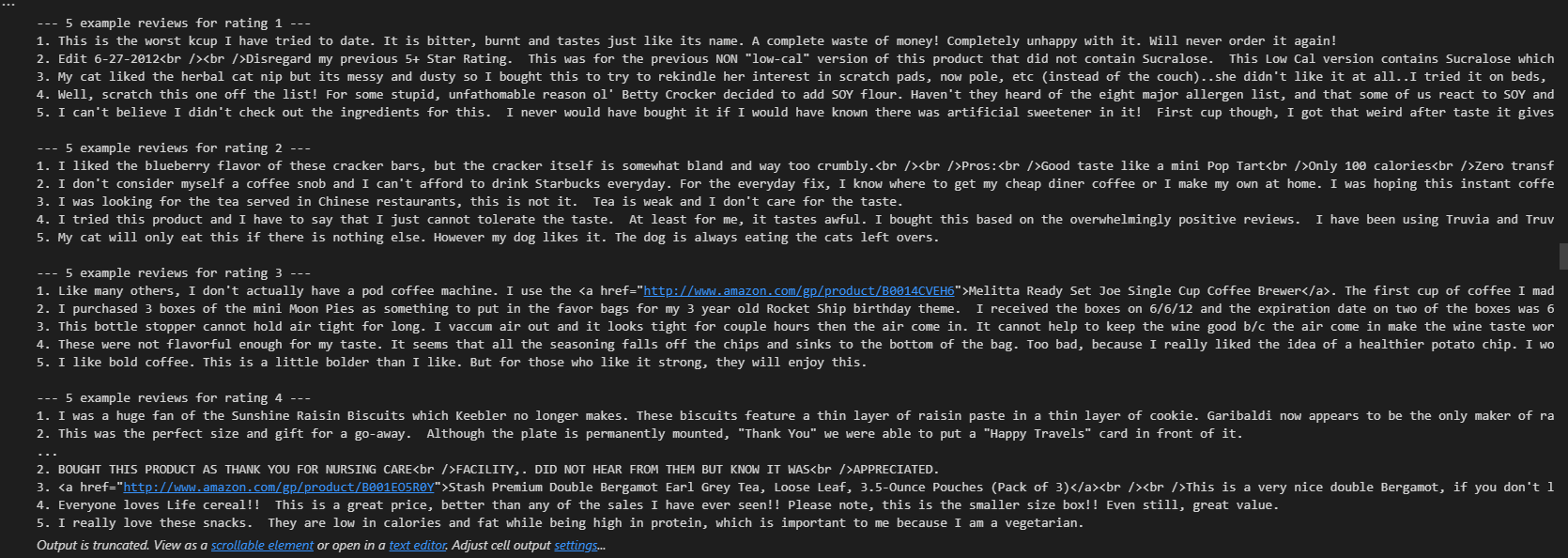
**5 Example Reviews per Rating**

for rating in sorted(df['Score'].unique()):

print(f"\n--- 5 sample reviews for rating {rating} ---")

for review in df[df['Score'] == rating]['review\_text'].sample(5, random\_state=42):

print('-', review)



**Imbalanced & Balanced Dataset Creation**

Imbalanced Dataset

total\_samples = 2000

target = {1: 0.10, 2: 0.15, 3: 0.20, 4: 0.25, 5: 0.30}

samples\_per\_class = {star: int(total\_samples \* pct) for star, pct in target.items()}

dfs = []

for star, n in samples\_per\_class.items():

grp = df[df['Score'] == star]

dfs.append(grp.sample(n=n, random\_state=42, replace=len(grp) < n))

imbalanced = pd.concat(dfs).sample(frac=1, random\_state=42)

imbalanced.to\_csv('data/cleaned\_dataset/imbalanced\_data.csv', index=False)

**Balanced Dataset**

min\_n = df['Score'].value\_counts().min()

balanced = df.groupby('Score').sample(n=min\_n, random\_state=42)

balanced.to\_csv('data/cleaned\_dataset/balanced\_data.csv', index=False)

**Stopwords Removal**

import pandas as pd

file\_path = r'D:\Projects\automated-review-rating-system\data\cleaned\_dataset\imbalanced\_data.csv'

df = pd.read\_csv(r'D:\Projects\automated-review-rating-system\data\cleaned\_dataset\imbalanced\_data.csv')

from collections import Counter

import itertools

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

def get\_top\_words(texts, n=10):

all\_words = list(itertools.chain(\*[t.split() for t in texts]))

filtered\_words = [w for w in all\_words if w not in stop\_words]

return Counter(filtered\_words).most\_common(n)

def get\_removed\_stopwords(texts):

all\_words = list(itertools.chain(\*[t.split() for t in texts]))

removed\_words = [w for w in all\_words if w in stop\_words]

return Counter(removed\_words).most\_common()

# Top 10 words for each rating

for rating in sorted(df['Score'].unique()):

texts = df[df['Score'] == rating]['Text']

print(f"Top words for rating {rating}:")

print(get\_top\_words(texts, n=10))

print()

print(f"All removed stopwords for rating {rating}:")

removed\_sw = get\_removed\_stopwords(texts)

# optionally print top N removed stopwords, e.g., top 20

print(removed\_sw[:20])

print()



**Top words for rating 1:**

[('I', 521), ('/><br', 153), ('like', 86), ('The', 75), ('one', 58), ('product', 56), ('This', 56), ('would', 55), ('taste', 53), ('-', 47)]

**All removed stop words for rating 1:**

[('the', 720), ('and', 436), ('to', 413), ('a', 367), ('of', 324), ('it', 226), ('is', 215), ('this', 205), ('was', 197), ('that', 191), ('in', 185), ('for', 158), ('not', 136), ('my', 129), ('have', 127), ('with', 117), ('but', 114), ('are', 110), ('on', 103), ('they', 101)]

**Top words for rating 2:**

[('I', 940), ('like', 176), ('/><br', 162), ('The', 125), ('would', 118), ('It', 104), ('taste', 102), ('coffee', 94), ('product', 89), ('one', 86)]

**All removed stop words for rating 2:**

[('the', 1100), ('a', 696), ('and', 638), ('to', 636), ('of', 495), ('it', 399), ('is', 393), ('this', 333), ('in', 287), ('not', 285), ('was', 277), ('that', 269), ('but', 257), ('for', 251), ('have', 195), ('my', 194), ('with', 191), ('are', 170), ('you', 164), ('as', 155)]

**Top words for rating 3:**

[('I', 1590), ('/><br', 316), ('like', 297), ('The', 220), ('would', 209), ('taste', 181), ('It', 155), ('good', 142), ('one', 131), ('product', 130)]

**All removed stop words for rating 3:**

[('the', 2129), ('a', 1332), ('and', 1133), ('to', 1031), ('of', 943), ('it', 769), ('is', 727), ('this', 561), ('in', 535), ('that', 520), ('but', 512), ('for', 497), ('not', 444), ('was', 425), ('with', 393), ('have', 354), ('my', 337), ('you', 310), ('are', 290), ('as', 272)]

**Top words for rating 4:**

[('I', 1699), ('/><br', 372), ('like', 342), ('The', 258), ('good', 235), ('one', 180), ('coffee', 171), ('taste', 161), ('It', 158), ('would', 151)]

**All removed stopwords for rating 4:**

[('the', 2055), ('a', 1589), ('and', 1385), ('to', 1223), ('of', 1037), ('is', 954), ('it', 853), ('for', 643), ('in', 593), ('this', 569), ('that', 563), ('but', 493), ('with', 433), ('not', 391), ('are', 387), ('my', 385), ('have', 367), ('was', 361), ('as', 345), ('you', 328)]

**Top words for rating 5:**

[('I', 841), ('/><br', 139), ('love', 122), ('like', 121), ('The', 107), ('great', 101), ('one', 98), ('It', 95), ('This', 90), ('good', 76)]

**All removed stopwords for rating 5:**

[('the', 913), ('and', 838), ('a', 675), ('to', 573), ('is', 465), ('of', 464), ('it', 390), ('for', 347), ('this', 330), ('in', 314), ('that', 221), ('are', 216), ('have', 213), ('with', 213), ('my', 204), ('you', 195), ('on', 168), ('but', 168), ('as', 154), ('so', 151)]

**Lemmatization**

def lemmatize\_text(text):

"""

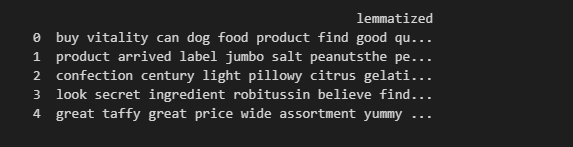
Lemmatize text using SpaCy and remove stopwords.

"""

doc = nlp(str(text))

tokens = [token.lemma\_ for token in doc if not token.is\_stop and token.is\_alpha]

return ' '.join(tokens)



**Filtering reviews (Less than 3 words or excessive words)**

def filter\_reviews(df, text\_column='lemmatized', min\_words=3, max\_words=200):

"""

Filters the DataFrame to keep only rows where the `text\_column` has

at least `min\_words` and at most `max\_words` words.

"""

df['word\_count'] = df[text\_column].apply(lambda x: len(str(x).split()))

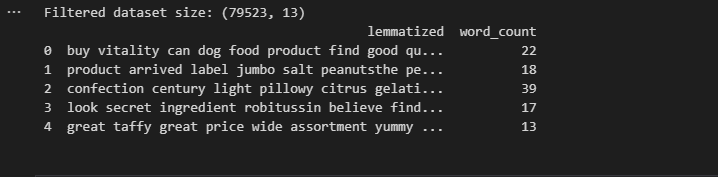
filtered\_df = df[(df['word\_count'] >= min\_words) & (df['word\_count'] <= max\_words)].reset\_index(drop=True)

return filtered\_df

filtered\_df = filter\_reviews(df, text\_column='lemmatized', min\_words=3, max\_words=200)

print(f"Filtered dataset size: {filtered\_df.shape}")

print(filtered\_df[[ 'lemmatized', 'word\_count']].head())



**Data Cleaning - Removing URLs, Tags, Emojis, Punctuation, Special Characters**

import pandas as pd

import os

import re

def clean\_text(text):

"""

Remove URLs, HTML tags, emojis, punctuation, and special characters from text.

"""

text = str(text).lower() # Lowercase

text = re.sub(r'http\S+|www.\S+', '', text) # Remove URLs

text = re.sub(r'<.\*?>', '', text) # Remove HTML tags

text = re.sub(r'[^\w\s]', '', text) # Remove emojis, punctuation, special chars

text = re.sub(r'\s+', ' ', text).strip() # Remove extra whitespace

return text

folder\_path = r"D:\Projects\automated-review-rating-system\data\Day3 Data"

df\_list = []

for filename in os.listdir(folder\_path):

if filename.endswith('.csv'):

filepath = os.path.join(folder\_path, filename)

df\_temp = pd.read\_csv(filepath)

if 'Text' in df\_temp.columns:

df\_temp['clean\_text'] = df\_temp['Text'].apply(clean\_text)

df\_list.append(df\_temp)

else:

print(f"Warning: 'Text' column missing in {filename}")

# Combine all cleaned DataFrames

df\_all\_cleaned = pd.concat(df\_list, ignore\_index=True)

print(f"Combined dataset size after cleaning: {df\_all\_cleaned.shape}")

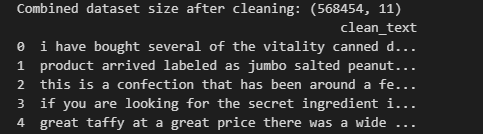
print(df\_all\_cleaned[['clean\_text']].head())

# Saving cleaned DataFrame to a CSV file

output\_path = r"D:\Projects\automated-review-rating-system\data\Day3 Data\cleaned\_reviews.csv"

df\_all\_cleaned.to\_csv(output\_path, index=False)

print(f"Cleaned data saved to { automated-review-rating-system\data\cleaned\_dataset\cleaned\_dataset.csv}")



**Data Visualization : Bar chart for review count per rating**

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(8,5))

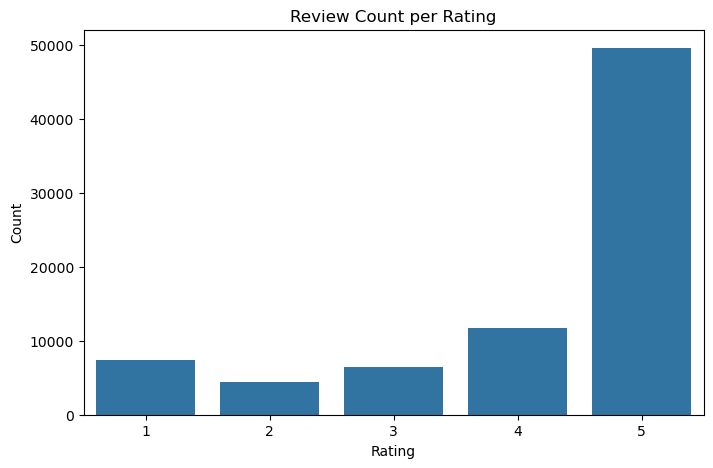
sns.countplot(x='Score', data=filtered\_df)

plt.title('Review Count per Rating')

plt.xlabel('Rating')

plt.ylabel('Count')

plt.show()



**Boxplot for word count distribution by rating**

plt.figure(figsize=(8,5))

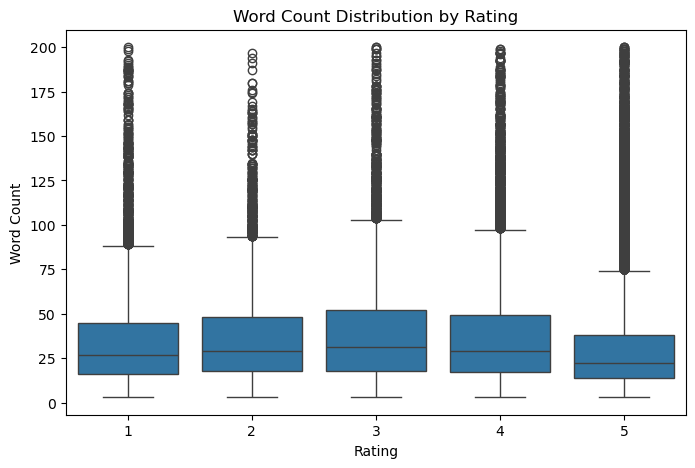
sns.boxplot(x='Score', y='word\_count', data=filtered\_df)

plt.title('Word Count Distribution by Rating')

plt.xlabel('Rating')

plt.ylabel('Word Count')

plt.show()



**Displaying sample reviews for each rating**

for Score in sorted(filtered\_df['Score'].unique()):

print(f"\n--- {Score} Star Reviews ---")

samples = filtered\_df[filtered\_df['Score']==Score]['Text'].head(3)

for review in samples:

print(review)

**Creating a balanced dataset : Sample equal number of reviews from each rating class to balance the dataset**

import pandas as pd

# Use your processed DataFrame, e.g., filtered\_df

# Set desired sample count per class

samples\_per\_class = 2000

# Sample equal number from each rating (with or without replacement as needed)

balanced\_df = filtered\_df.groupby('Score').apply(

lambda x: x.sample(n=samples\_per\_class, random\_state=42)

if len(x) >= samples\_per\_class else x.sample(n=samples\_per\_class, replace=True, random\_state=42)

)

# Remove multi-index from groupby

balanced\_df = balanced\_df.reset\_index(drop=True)

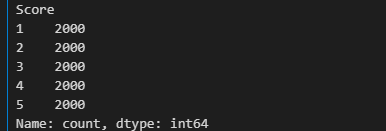
# Check balance

print(balanced\_df['Score'].value\_counts().sort\_index())

output\_path = r"D:\Projects\automated-review-rating-system\data\Day3 Data\balanced\_reviews.csv"

balanced\_df.to\_csv(output\_path, index=False)

print(f"Balanced dataset saved to {output\_path}")



**Train Test split for balanced dataset**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

# Load the balanced dataset

file\_path = r"D:\Projects\automated-review-rating-system\data\Day3 Data\balanced\_reviews.csv"

balanced\_df = pd.read\_csv(file\_path)

# Separate features and target

X = balanced\_df['clean\_text'] # replace if using another text column like 'lemmatized'

y = balanced\_df['Score']

# Stratified train-test split, 80% train, 20% test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, stratify=y, random\_state=42, shuffle=True)

print(f"Train set size: {len(X\_train)}, Test set size: {len(X\_test)}")

print(f"Train class distribution:\n{y\_train.value\_counts()}")

print(f"Test class distribution:\n{y\_test.value\_counts()}")

# TF-IDF Vectorization: fit on train, transform on both sets

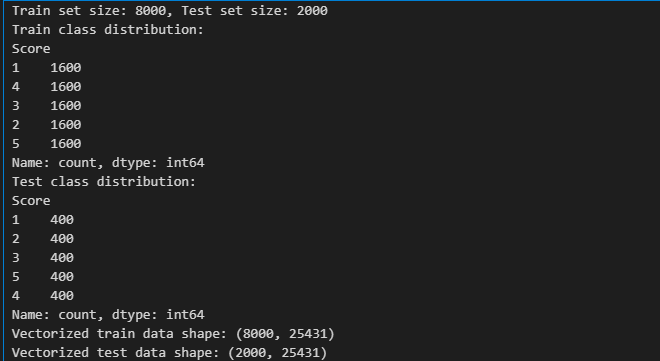
vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

print(f"Vectorized train data shape: {X\_train\_vec.shape}")

print(f"Vectorized test data shape: {X\_test\_vec.shape}")



**Train Test Split for imbalanced data**

from sklearn.model\_selection import train\_test\_split

X = df['lemmatized']

y = df['Score']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, stratify=y, random\_state=42, shuffle=True

)

print("Train and test sizes:", len(X\_train), len(X\_test))

print("Train class distribution:\n", y\_train.value\_counts())

print("Test class distribution:\n", y\_test.value\_counts())

#Vectorization

from sklearn.feature\_extraction.text import TfidfVectorizer

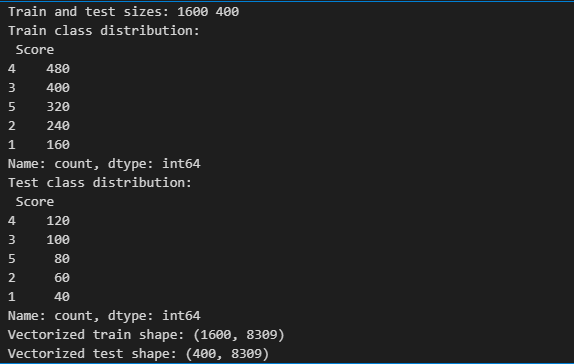
vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train) # Fit only on train!

X\_test\_vec = vectorizer.transform(X\_test)

print("Vectorized train shape:", X\_train\_vec.shape)

print("Vectorized test shape:", X\_test\_vec.shape)



**Algorithm Deep Dive Documentation: Logistic Regression**

**A. Algorithm Type & Objective**

Type: Linear, supervised classification (can also be used for regression, but here for text classification).

Objective: Directly predicts the probability that an input text belongs to each possible class (e.g., a sentiment rating), using a logistic (“sigmoid” or “softmax” in multiclass) function.

**B. How the Algorithm Works (Mathematical Intuition)**

**Features**: Each document is represented as a vector (e.g., TF-IDF features from vectorization).

**Linear combination**: Calculates a weighted sum of these features, plus a bias term**.**

**Logistic function:**Applies a sigmoid (binary case) or softmax (multiclass) to output a probability for each class.

**Training:**Finds the optimal set of weights that minimize the difference between predicted and actual labels (via cross-entropy/log-loss).

**Decision boundary**: Finds the “best” linear separation between classes in the high-dimensional feature space.

**Mathematics:**

**p(y=1∣x)=σ(w⋅x+b)=11+e−(w⋅x+b)p(y=1∣x)=σ(w⋅x+b)=1+e−(w⋅x+b)1**

Where σσ is the sigmoid, ⋅⋅ is the dot product, and ww, bb are learned.

**C. Key Hyper parameters**

• Inverse regularization strength (C=1/lambda). Lower values mean stronger regularization (simpler model), higher values allow a better fit (risking overfit).

Default: 1.0. Tune over log-scale, e.g., [0.01, 0.1, 1, 10, 100].

• **Penalty**: Type of regularization, e.g., 'l2' (default, recommended), or 'l1' (can enforce sparsity).

• **Solver**: Optimization algorithm. 'liblinear' (good for small datasets and L1), 'lbfgs' (default, fits multiclass well).

• **max\_iter**: Maximum solver iterations. Increase if convergence warnings arise.

• **class\_weight**: Handle class imbalance ('balanced' sets weights automatically).

**Strengths & Limitations (for Text Classification)**

**Strengths:**

Simple, interpretable, and fast to train—even with many features.

Handles large, sparse feature sets elegantly (ideal for bag-of-words, TF-IDF, one-hot).

Outputs well-calibrated probabilities.

Robust to correlated features (with regularization).

**Limitations:**

Assumes linear separability—may underperform with complex, nonlinear phenomena.

Each feature (word/ngram) contributes linearly; cannot capture word sequences or non-obvious feature interactions as well as tree-based or neural models.

Sensitive to outliers and collinearity if regularization is not used.

**When to Use / When Not to Use**

Use When:

You want a strong baseline for text classification.

Datasets are medium to large, features are high-dimensional and sparse.

Fast training and model transparency are important.

Multi-class or multi-label problems with interpretable weights.

Avoid When:

Underlying relationships are highly nonlinear or need sequence/context modelling.

You need to capture subtle patterns or word order (consider RNNs, CNNs, transformers for these).

Data is extremely imbalanced and cannot be rectified through sampling/weighting.