AUTOMATED REVIEW RATING SYSTEM

1. Project Overview

A system that analyzes customer reviews and predicts ratings.The goal is to extract meaningful features from raw review data and train a model that can accurately estimate the star rating, enabling scalable sentiment analysis and product feedback evaluation.

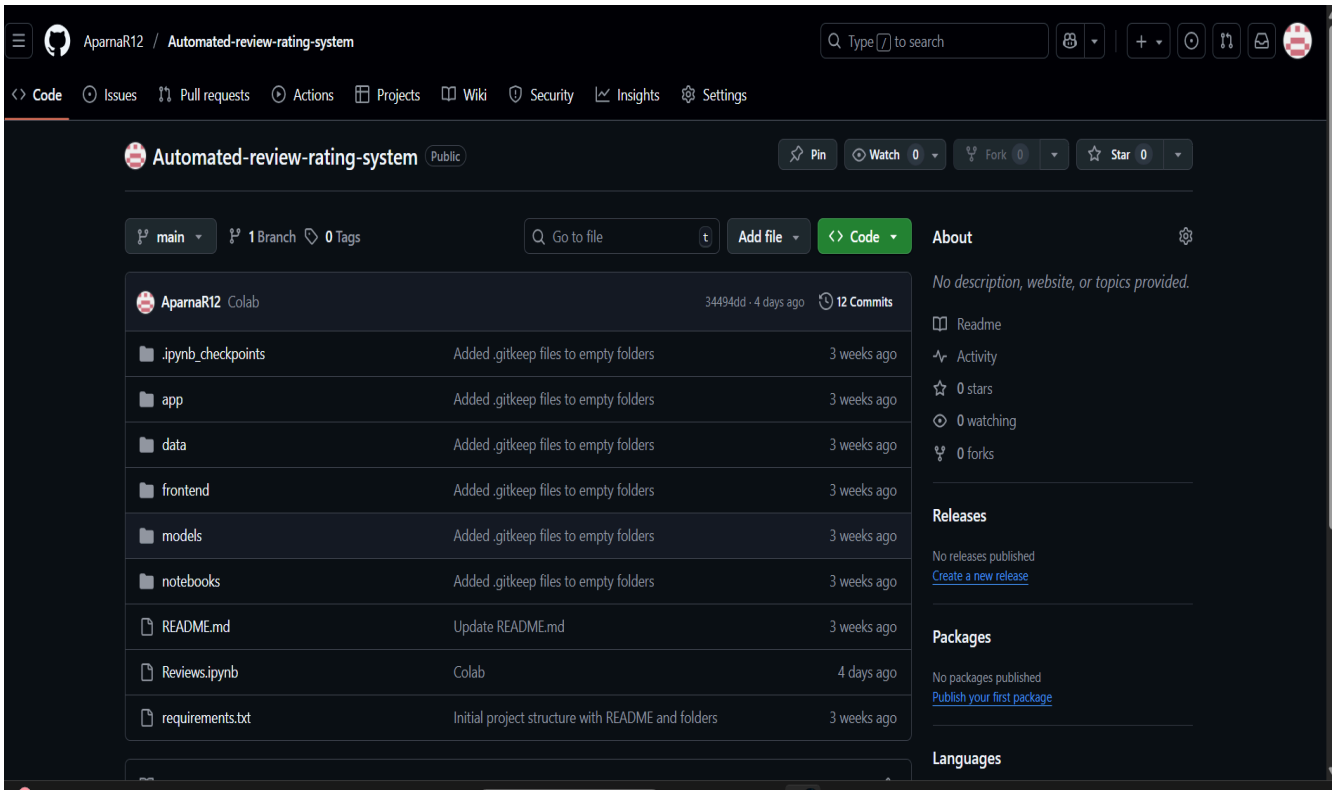
2. Environment setup

It was developed and executed in **Google Colab**, a cloud-based Jupyter notebook environment that supports GPU acceleration and seamless Python workflows.The implementation was carried out using Python 3.10.
The following libraries are used for the implementation:

- Pandas
- Seaborn
- Numpy
- Matplotlib
- Scikit Learn
- Re
- NLTK

3. Github Project Setup

A Github repository named Automated-rating-review-system created to manage the project code and resources. The repository has a clear, well-structured layout, which aligns with the folder structure to provide clarity to codebase.The repository contains necessary collaboration capabilities,version control and localization for all project files.



4. Dataset Collection

The Automated Rating Review System is built upon a publicly available collection of customer feedback from Kaggle. Downloaded a pre-cleaned dataset containing review text and ratings. The dataset was downloaded in CSV format. The data utilized for training and evaluation of this system is sourced from 3 publicly available dataset of amazon reviews. It can be accessed through these links :[*Amazon Fine Food Reviews Kaggle*,*https://www.kaggle.com/datasets/dongrelaxman/amazon-reviews-dataset*,https://www.kaggle.com/datasets/gzdekzlkaya/amazon-product-reviews-dataset].This dataset contains reviews and ratings which provides the text of the review and a corresponding 1 to 5 star rating, which were crucial for developing the sentiment classification model.

5. Dataset Description

It utilizes three Amazon review datasets, which is used for training and evaluating the Automated Review Rating System. All datasets were sourced from Kaggle and manually curated in Excel for preprocessing and analysis. We need only ratings and review text for training so dropped other columns for efficient evaluation.

Dataset 1 [Amazon reviews product dataset]

This dataset contains user-generated reviews from Amazon customers. Each entry includes detailed metadata about the reviewer, such as their ID, name, and helpfulness votes, along with the review title, full text, and a numeric rating. Timestamp information is provided in both Unix and human-readable formats. This dataset is ideal for sentiment analysis, review quality assessment, and modeling user feedback behavior.

	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime	day_diff	helpful_yes	total_vote
0	A3SBTW3WS4IQSN	B007WTAJTO	NaN	[0, 0]	No issues.	4	Four Stars	1406073600	7/23/2014	138	0	0
1	A18K1ODH1I2MVB	B007WTAJTO	0mie	[0, 0]	Purchased this for my device, it worked as adv...	5	MOAR SPACE!!!	1382659200	10/25/2013	409	0	0
2	A2FII3I2MBMUIA	B007WTAJTO	1K3	[0, 0]	it works as expected. I should have sprung for...	4	nothing to really say....	1356220800	12/23/2012	715	0	0
3	A3H99DFEG68SR	B007WTAJTO	1m2	[0, 0]	This think has worked out great.Had a diff. br...	5	Great buy at this price!!! *** UPDATE	1384992000	11/21/2013	382	0	0
4	A375ZM4U047O79	B007WTAJTO	2&1/2Men	[0, 0]	Bought it with Retail Packaging, arrived legit...	5	best deal around	1373673600	7/13/2013	513	0	0

Dataset 2 [Amazon reviews Dataset]

This dataset contains user-generated reviews from international customers evaluating exam-related profiles. Each entry includes detailed metadata about the reviewer, such as their name, country, and total review count, along with the review title, full text, and a numeric rating. It also includes profile links and exam dates for contextual analysis. This dataset is ideal for sentiment analysis, fraud detection, and behavioral modeling in educational review systems.

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Dataset 3 [Amazon food reviews]

This dataset contains user-generated reviews from food product customers. Each entry includes detailed metadata about the reviewer, such as their profile name, helpfulness votes, and review timestamp, along with the review title, full text, and a numeric rating. It also includes product identifiers for linking reviews to specific items. This dataset is ideal for sentiment analysis, product feedback evaluation, and modeling consumer behavior.

	Reviewer Name	Profile Link	Country	Review Count	Review Date	Rating	Review Title	Review Text	Date of Experience
0	Eugene ath	/users/66e8185ff1598352d6b3701a	US	1 review	2024-09-16T13:44:26.000Z	Rated 1 out of 5 stars	A Store That Doesn't Want to Sell Anything	I registered on the website, tried to order a ...	16-Sep-24
1	Daniel ohalloran	/users/5d75e460200c1f6a6373648c	GB	9 reviews	2024-09-16T18:26:46.000Z	Rated 1 out of 5 stars	Had multiple orders one turned up and...	Had multiple orders one turned up and driver h...	16-Sep-24
2	p fisher	/users/546cfcf1000064000197b88f	GB	90 reviews	2024-09-16T21:47:39.000Z	Rated 1 out of 5 stars	I informed these reprobates	I informed these reprobates that I WOULD NOT B...	16-Sep-24
3	Greg Dunn	/users/62c35cdbacc0ea0012ccaffa	AU	5 reviews	2024-09-17T07:15:49.000Z	Rated 1 out of 5 stars	Advertise one price then increase it on website	I have bought from Amazon before and no proble...	17-Sep-24
4	Sheila Hannah	/users/5ddbe429478d88251550610e	GB	8 reviews	2024-09-16T18:37:17.000Z	Rated 1 out of 5 stars	If I could give a lower rate I would	If I could give a lower rate I would! I cancel...	16-Sep-24

6. Data Preprocessing

Handling missing data

It is a quick way to audit missing values in your dataset.Also dropped the rows containing missing values.This is useful when missing values are expected or can be reasonably imputed. Drop command can be used if the missing data is minimal and won’t affect analysis.

Code :

```
df.isnull().sum()

df.dropna(inplace=True)
```

Removal of duplicates

It checks for duplicate rows in your DataFrame. It returns a Boolean Series (True for duplicates), and .sum() counts how many True values are there. It can quantify how many rows are exact duplicates.This removes all rows that are exact duplicates of earlier rows. Removing duplicates is essential for preventing bias in models, improving efficiency in training and analysis,ensuring data integrity when merging or aggregating.

Code :

```
df.duplicated().sum()

df.drop_duplicates(inplace=True)
```

Removal of exact duplicates

These are rows where both the review text and the rating are identical.

Code :

```
exact_duplicates =
merged_df[merged_df.duplicated(subset=['Reviews','Rating'], keep=False)]

print(exact_duplicates)
```

Removal of conflicting reviews

These are rows where the review text is identical, but the rating differs.

Code :

```
conflicting_reviews = merged_df[merged_df.duplicated(subset=['Reviews'],
keep=False)]

conflicting_reviews = conflicting_reviews.groupby('Reviews').filter(lambda
x: len(x['Rating'].unique()) > 1)

print(conflicting_reviews)
```

Nan Values Removal

Removes all rows where the column `Rating` has a missing (NaN) value.
In sentiment or rating prediction tasks, every review must have a corresponding rating. Missing ratings make the data unusable for supervised learning (because there's no label to train on).

Code :

```
merged_df = merged_df.dropna(subset=['Rating'])

merged_df = merged_df.reset_index(drop=True)

print(f"Dataset after dropping NaN ratings: {merged_df.shape}")
```

Convert String to Numeric

This code ensures that the Rating column contains clean, numerical values that can be used for statistical analysis or model training.

Code :

```
merged_df['Rating'] =
merged_df['Rating'].astype(str).str.extract(r'(\d+)').astype(float)
```

Lowercase Conversion

All review text was converted to lowercase using Python string methods to ensure uniformity and reduce token duplication (e.g., "Great" vs. "great").Converting to lowercase reduces noise in your data,improves model generalization and makes downstream tasks like stopword removal and lemmatization more consistent.

Code :

```
text = text.lower()
```

Remove URLs, HTML Tags, Punctuation, and Special Characters

Regular expressions (re module) were used to strip out:

- URLs (http://, www. etc.)
- HTML tags (
,)

- Punctuation and special symbols

Code :

```
import re

def clean_text(text):

if not isinstance(text, str):

return None

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

    text = re.sub(r"<.*?>", " ", text)

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

    text = re.sub(r"\d+", " ", text)

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

    # Remove emojis (common ranges)

    emoji_pattern = re.compile(

        "[" +

        "\U0001F600-\U0001F64F" +  # emoticons

        "\U0001F300-\U0001F5FF" +  # symbols & pictographs

        "\U0001F680-\U0001F6FF" +  # transport & map symbols

        "\U0001F1E0-\U0001F1FF" +  # flags

        "]+", flags=re.UNICODE

    )

    text = emoji_pattern.sub("", text)
```

Remove Stopwords (Using NLTK)

Stopwords are common words (e.g., “the”, “is”, “and”) that carry little semantic weight. These were removed using NLTK’s English stopword list.

- Stopword Count: 198 words.

Code :

```
import nltk

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

# Print total number and the stopwords themselves

print(f"Total stopwords: {len(stop_words)}\n")

print("Stopwords list:\n")

print(sorted(stop_words))
```

Output :

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all',
'am', 'an', 'and', 'any', 'are', 'aren', "aren't", 'as', 'at', 'be',
'because', 'been', 'before', 'being', 'below', 'between', 'both',
'but', 'by', 'can', 'couldn', "couldn't", 'd', 'did', 'didn',
"didn't", 'do', 'does', 'doesn', "doesn't", 'doing', 'don', "don't",
'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had',
'hadn', "hadn't", 'has', 'hasn', "hasn't", 'have', 'haven',
"haven't", 'having', 'he', "he'd", "he'll", "he's", 'her', 'here',
'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', "i'd",
"i'll", "i'm", "i've", 'if', 'in', 'into', 'is', 'isn', "isn't",
'it', "it'd", "it'll", "it's", 'its', 'itself', 'just', 'll', 'm',
'ma', 'me', 'mightn', "mightn't", 'more', 'most', 'mustn',
"mustn't", 'my', 'myself', 'needn', "needn't", 'no', 'nor', 'not',
'now', 'o', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'our',
'ours', 'ourselves', 'out', 'over', 'own', 're', 's', 'same',
'shan', "shan't", 'she', "she'd", "she'll", "she's", 'should',
"should've", 'shouldn', "shouldn't", 'so', 'some', 'such', 't',
'than', 'that', "that'll", 'the', 'their', 'theirs', 'them',
'themselves', 'then', 'there', 'these', 'they', "they'd", "they'll",
"they're", "they've", 'this', 'those', 'through', 'to', 'too',
'under', 'until', 'up', 've', 'very', 'was', 'wasn', "wasn't", 'we',
"we'd", "we'll", "we're", "we've", 'were', 'weren', "weren't",
'what', 'when', 'where', 'which', 'while', 'who', 'whom', 'why',
'will', 'with', 'won', "won't", 'wouldn', "wouldn't", 'y', 'you',
"you'd", "you'll", "you're", "you've", 'your', 'yours', 'yourself',
'yourselves']
```

Lemmatization

Lemmatization reduces words to their base form (e.g., “running” → “run”) while preserving meaning. It was performed using NLTK’s WordNetLemmatizer.

Code :

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
# Tagger name changed in newer NLTK; try both
try:
    nltk.download('averaged_perceptron_tagger', quiet=True)
except:
    pass
try:
    nltk.download('averaged_perceptron_tagger_eng', quiet=True)
except:
    pass
# Lemmatization
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
def _to_wn_pos(tag: str):
    """Map NLTK POS tags to WordNet POS tags."""
    if tag.startswith('J'):
        return wn.ADJ
    if tag.startswith('V'):
        return wn.VERB
    if tag.startswith('N'):
        return wn.NOUN
    if tag.startswith('R'):
        return wn.ADV
    return wn.NOUN  # fallback
```

Why Lemmatization is better than Stemming

Lemmatization is generally preferred over stemming, especially for production-grade or semantically sensitive tasks.

- **Semantic clarity**: Lemmatization returns actual words, which improves interpretability and downstream tasks like topic modeling or sentiment analysis. **Contextual accuracy**: It considers the part of speech. For example, "better" becomes "good" only if tagged as an adjective.
- **Reduced noise**: Stemming can over-aggressively chop words, leading to ambiguity. For instance:
    - "Studies" → "studi" (stemmed)
    - "Studies" → "study" (lemmatized)
- **Better for embeddings**: If you're using word vectors or transformer models, lemmatized tokens align better with pretrained vocabularies.

Filter Out Short and Excessively Long Reviews

To maintain quality and relevance:

- Reviews with fewer than 3 words were removed.
- Reviews with excessive length >200 words were filtered out.

Code :

```
if len(lemmas) < 3 or len(lemmas) > 200:

        return ""

    return " ".join(lemmas)
```

7. Balancing Datasets

It contains a sample equal number of reviews i.e. 2000 reviews from each rating class. It ensures balance across all 5 classes (1–5 stars).It prepares the text for test and train split. Balancing ensures that your model learns equally from all rating levels, improving generalization and fairness.

Code :

```
from sklearn.utils import resample
```

```
import pandas as pd

# Ensure Rating column is numeric

merged_df['Rating'] = pd.to_numeric(merged_df['Rating'], errors='coerce')

merged_df = merged_df.dropna(subset=['Rating'])

merged_df['Rating'] = merged_df['Rating'].astype(int)

# Show class distribution before balancing

print("Before balancing:")

print(merged_df['Rating'].value_counts())

# Define desired number of samples per class

samples_per_class = 2000

# Balance dataset by sampling equal number of reviews for each rating

balanced_df = (

    merged_df.groupby('Rating', group_keys=False)

    .apply(lambda x: x.sample(n=min(len(x), samples_per_class),
random_state=42))

    .reset_index(drop=True)

)

# Show results

print("\nAfter balancing:")

print(balanced_df['Rating'].value_counts())

print("Balanced dataset shape:", balanced_df.shape)
```

Output :

```
Balanced dataset shape: (10000, 3)
```

```
Rating
1.0    2000
2.0    2000
3.0    2000
4.0    2000
5.0    2000
Name: count, dtype: int64
```

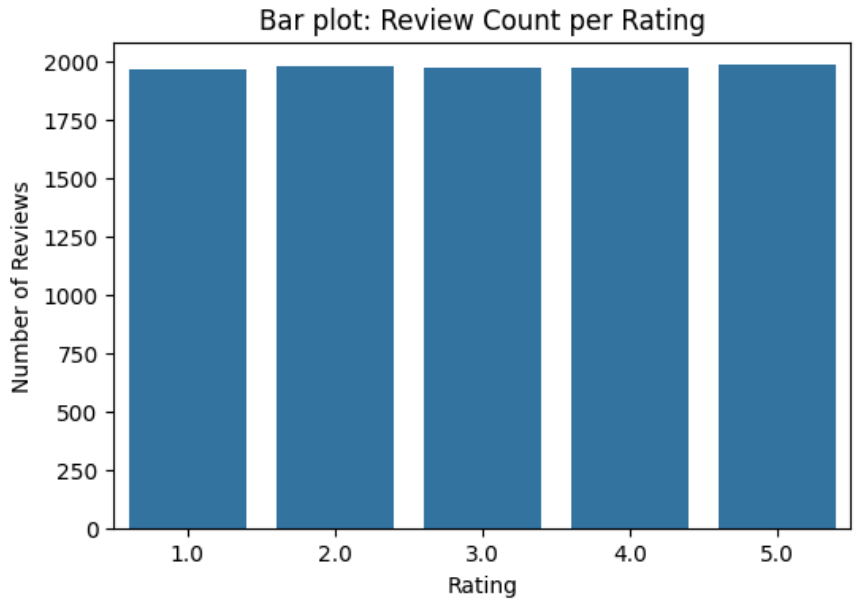
8. Visualization

Bar Plot

A bar plot of review count per rating is a simple visualization that shows how many reviews fall into each rating category.

Code :

```
plt.figure(figsize=(6,4))

sns.countplot(x="Rating", data=balanced_df,
order=sorted(balanced_df["Rating"].unique()))

plt.title("Bar plot: Review Count per Rating")

plt.xlabel("Rating")

plt.ylabel("Number of Reviews")

plt.show()
```

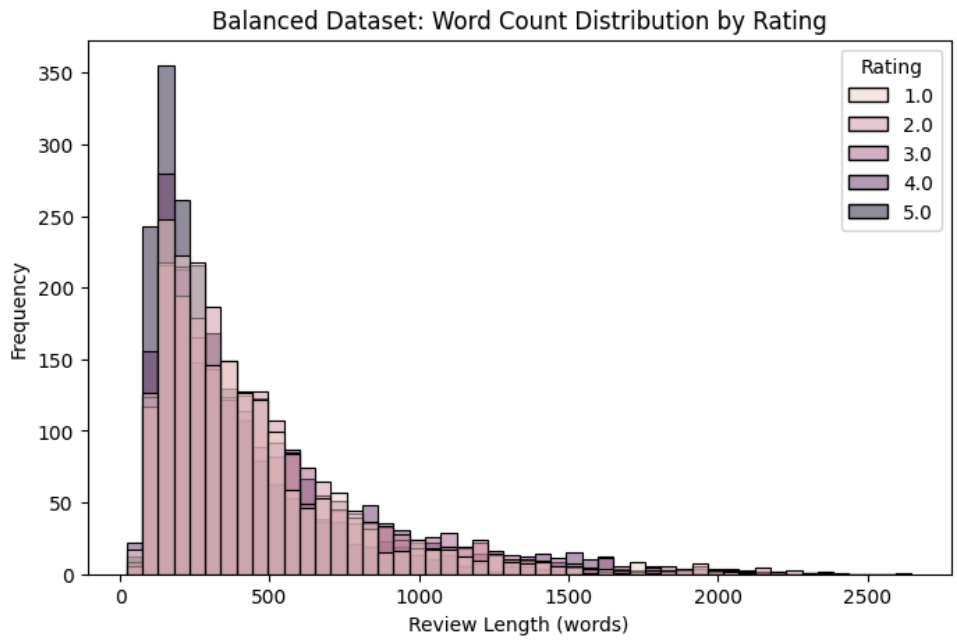


Histogram

A histogram of word count distribution by rating is a powerful way to visualize how the length of text (typically reviews or comments) varies across different rating levels.

Code :

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

sns.histplot(data=balanced_df, x="review_length", hue="Rating", bins=50,
kde=False)

plt.title("Balanced Dataset: Word Count Distribution by Rating")

plt.xlabel("Review Length (words)")

plt.ylabel("Frequency")

plt.show()
```

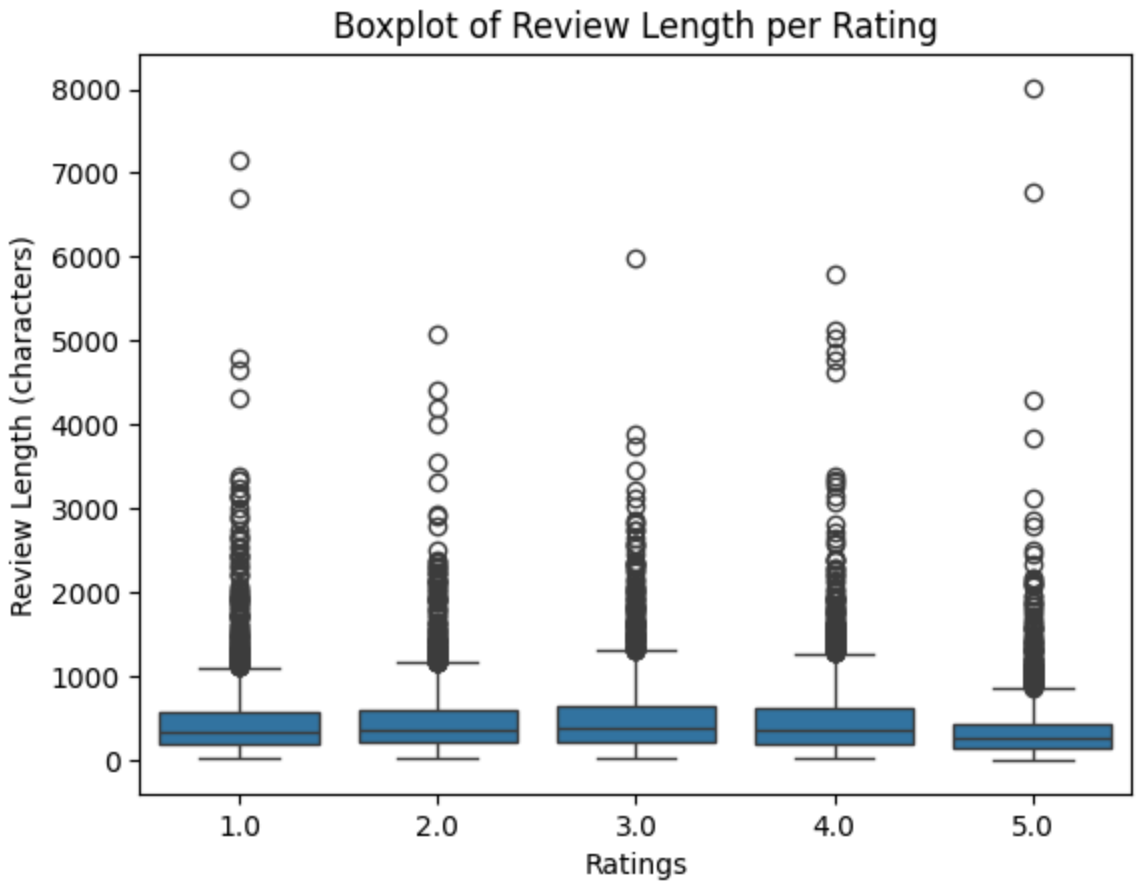


Box Plot

Box plot is a statistical visualization that shows the distribution of review length across different rating categories.

Code :

```
review_lengths = balanced_df['Reviews'].astype(str).apply(len)

sns.boxplot(x=balanced_df['Rating'], y=review_lengths)

plt.xlabel("Ratings")

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

plt.title("Boxplot of Review Length per Rating")

plt.show()
```

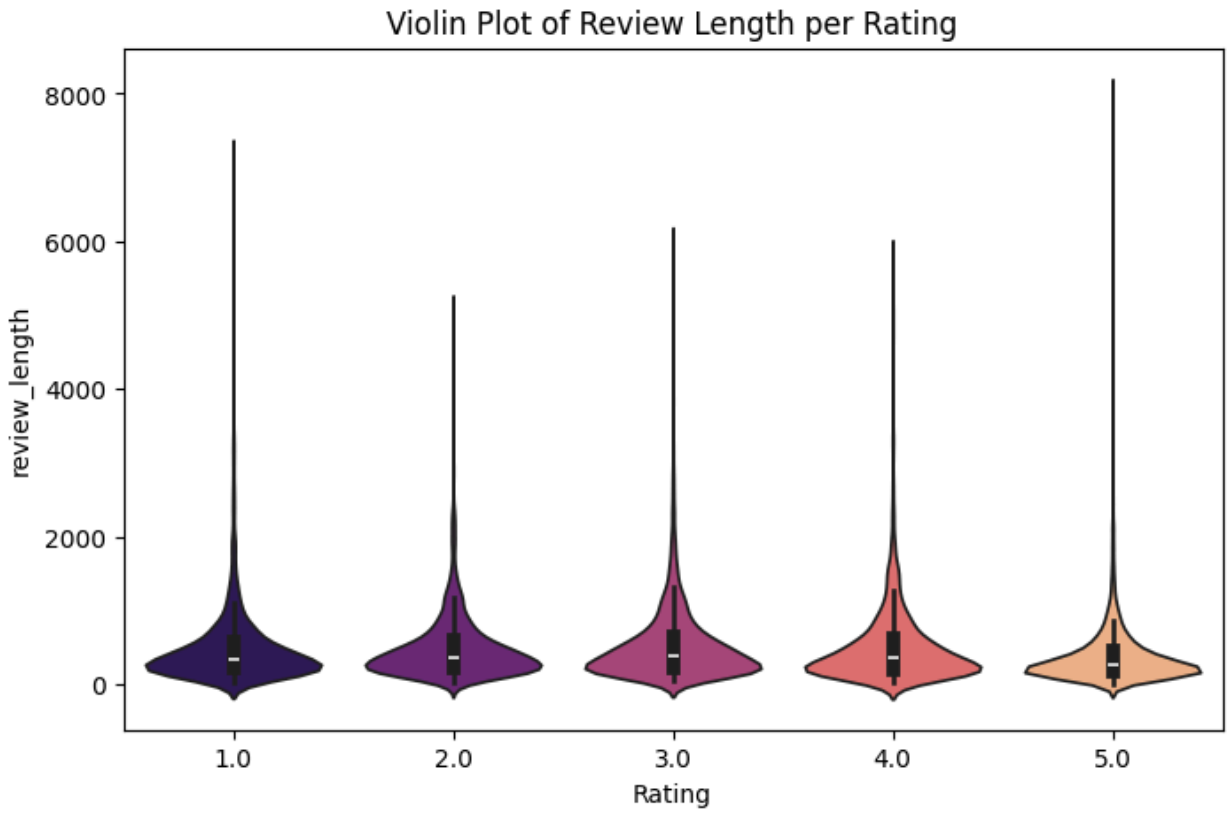


Violin Plot

A violin plot is a powerful visualization that combines a box plot and a kernel density plot to show the distribution of review length — across different rating categories.

Code :

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

sns.violinplot(x='Rating', y='review_length', data=balanced_df,
palette='magma')

plt.title('Violin Plot of Review Length per Rating')

plt.show()
```



Display 5 Sample Reviews Per Rating

It displays sample reviews from each rating(1-5 stars).

Code :

```
for rating in sorted(balanced_df["Rating"].unique()):

    print(f"\n--- Rating {rating} ---")

    samples = balanced_df[balanced_df["Rating"] ==
rating]["Reviews"].sample(n=5, random_state=42)

    for i, review in enumerate(samples, 1):

        print(f"{i}. {review}")
```

Output :

--- Rating 1 ---

1. amazon seller everything proof item still amazon attitude accept proof seller also jealous account top first time amazon make thing difficult

2. use thing deliver every house mine order cancel apparent reason really misleading description good

3. live ireland order amazon uk always slow despatch normally take week send stock give false delivery date always track parcel never anywhere near u suppose delivery date send quicker stop lying

4. awful company consistently ship incorrect defective product unable resolve issue contact person even cancel account bad company planet would rather shop twenty different website stuff need rip break robot anymore

5. amason prime member get tedious rug pull stream content pay free prime member late delivery day even though opted next day prime delivery earlier delivery week find harder justify money im pay pathetic service seriously consider cancel sue breach contract basos think im pay extra leave

--- Rating 2 ---

1. read review buy pack cranberry orange sugar biscotti expect crisp light others thick like biscuit child use teethe yes taste good certainly one make regular biscotti tasty crispy still omit sugar take looong dunk get soft go nonni biscotti day

2. item basket collectively worth quite sure expect buy glove something gummi worm cent stale cooky seed amazon tea way overprice deliver

3. order large version field europe orange rise version flower close really want pink rose want pink rose would order bouquet instead hop something really pop summer springy love bouquet price send order make hope best ordered lot flower flower think find someone else disappoint something cost would come look closer want

4. get husband drank pod yet use anymore think coffee weak less water recommend thanks free sample fun try new thing best coffee yet worth try update cut five packet put five cup filter full amount water coffee brew perfectly taste really good try instead use one shot packet work

5. amazon need stop lying people horrible company never follow thru sell crap

--- Rating 3 ---

1. sample kali ginger lemon drink mix cold wintry day throat bit inflamed sore drank hot direct foil pouch honest first sip burn quite bit go due ginger continue drink lemon honey flavor come fore finishing whole cup throat felt little bit well realize overall liked beverage say big question like well enough order honest answer probably gold kali ginger lemon one product might enjoy rare occasion one tempt stock make part regular diet definitely group consumer spicy drink appeal heartily recommend man enjoy traditional warm cold beverage let say really cup tea

2. ok son eat love carrot eats go move onto stage food

3. quality brand good really enjoyed three six three bland three like little less spicy would eat make mass buy pack kidney bean curry spice like great

4. nice perk senior citizen buy item receive free small drink ask counter clerk substitute limeade obviously would pay difference say limeade would still remain full price ask limeade labor intensive say question suggestion charge cent difference even extra cent fair ingredient cost full price insult customer sale clerk apologize say make sense either

5. get chef michael grill sirloin dry dog food mixed half half dog current food avoid get run could definitely tell like new food since seem pick old food eat costco brand salmon grain free food year huge change since second third ingredient type soy product however disturbing part oil realize several piece kibble laid pantry several day vacuum oil stain slate floor let say costco kibble stick old brand dog food dog appreciate little variety diet last week

--- Rating 4 ---

1. use site everytime recomand

2. haribo sour cherry vibrantly color look like red cherry green stem green leaf attach piece inch long cute unfortunately sugary sweet sour taste like cherry flavor cough drop come red white box medicinal purpose whatsoever like flavor artificial cherry good smell good look candy kid birthday party definitely hit need sugary pick grab set really want something sour try another kind haribo candy add thought would beautiful cake decoration

3. good soda bread easy make also obtain one item gift

4. try powdered milk year worth new try taste fine plan use blender make easy fast great somewhere fresh milk hard find good add smoothy protein booster many us take one star packaging nearly useless open zipper plastic bag closure function well fine powdery substance track clog become un closable almost bag also envelope shape tend flatten make scoop powder difficult plan dump wide mouth canister container tight lid efficient packaging traveler however light weight take least possible space great product camp travel probably still need double bag ziplock bag open seal problem

5. cracker good bit sweet look sesame cracker go cheese egg salad salami etc flavor sour cream etc

--- Rating 5 ---

1. order year ago repeatedly order since everyone share love much also get right amazon order also mix coffee feel particularly naughty top whip cream delicious

2. feel cold come product work get rest need traditional medicinals wonderful product work proclaim

3. cant keep enough house kid drink like go style love cause easy make water packet best part sugar free

4. best thing ever find use give dog pills great idea

5. love amazon loyal amazon prime customer year amazon fair great refund return policy deliver fast smoothly sometimes day delivery take picture delivery notify package arrive amazon wide variety choose also large

```
variety price item get pay hundred order amazon year refund one box spring
dent puller
```

10. Train Test Split

It split the dataset into training and testing sets while preserving class balance, followed by preprocessing and vectorization of text data.

Dataset Shuffling

- Before splitting, the dataset is randomly shuffled to eliminate any ordering bias.
- This ensures that samples are distributed uniformly and prevents temporal or positional patterns from influencing the model.

Stratified Train-Test Split

- A stratified split is performed to divide the dataset into training and testing sets (typically 80% train, 20% test).
- Stratification ensures that the class distribution remains consistent across both sets, which is crucial for balanced model evaluation.

Example using `train_test_split` from `sklearn`:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(

X, y, test_size=0.2, stratify=y, random_state=42)
```

Why Stratified is used

Stratified sampling is used to ensure that train and test sets reflect the same class distribution as the original dataset. This is important when working with imbalanced data, like review ratings where 5-star reviews might dominate.

Without stratification:

- The test set might end up with mostly high ratings and very few low ones.
- This skews evaluation metrics and makes the model look better than it really is.
- It can also lead to poor generalization when deployed on real-world data.

With stratification:

- Each class (e.g., ratings 1–5) is proportionally represented in both training and testing sets.
- The model learns from all classes and is evaluated fairly.
- It improves robustness, especially for classification tasks.

Here, `stratify=y` ensures that the split maintains the same distribution of labels as y. Since the review rating system spans multiple classes, stratification helps preserve balance and fairness across your pipeline.

Preprocessing (Applied Separately)

- Preprocessing steps are applied independently to the training and testing sets.
- This avoids data leakage and ensures that transformations are learned only from the training data.
- Typical preprocessing includes:
    - Lowercasing
    - Stopword removal (using NLTK)
    - Lemmatization
    - Duplicate handling

Text Vectorization

- Text data is vectorized using either **TF-IDF**
- It is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.
- The vectorizer is fit only on the training set to prevent leakage of test data statistics.
- Both training and testing sets are then transformed using the fitted vectorizer.

TF = Term Frequency in document / Total Words in Document

IDF = $\log_2$ (Total Documents / documents with term)

Code :

```
import joblib

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

# Split data

X_bal = balanced_df["Reviews"]

y_bal = balanced_df["Rating"]

X_train_bal_raw, X_test_bal_raw, y_train_bal_raw, y_test_bal_raw =
train_test_split(

    X_bal, y_bal,

    test_size=0.2,

    random_state=42,

    stratify=y_bal

)

# --- Clean training and testing reviews ---

X_train_cleaned = X_train_bal_raw.apply(clean_review)

X_test_cleaned = X_test_bal_raw.apply(clean_review)

# --- Keep only valid (non-empty) rows ---

train_valid = X_train_cleaned.notna()

test_valid = X_test_cleaned.notna()

# --- Filter y accordingly (same valid indices) ---

X_train_bal = X_train_cleaned[train_valid].reset_index(drop=True)

y_train_bal = y_train_bal_raw[train_valid].reset_index(drop=True)

X_test_bal = X_test_cleaned[test_valid].reset_index(drop=True)

y_test_bal = y_test_bal_raw[test_valid].reset_index(drop=True)

# --- TF-IDF Vectorizer ---
```

```
vectorizer_bal = TfidfVectorizer(
    max_features=20000,      # more features → better coverage
    ngram_range=(1, 3),      # 1-gram, 2-gram, and 3-gram
    sublinear_tf=True,       # scales tf values
    stop_words='english'     # removes common English words
)
X_train_bal_vec = vectorizer_bal.fit_transform(X_train_bal)
X_test_bal_vec = vectorizer_bal.transform(X_test_bal)
# --- Save vectorizer ---
joblib.dump(vectorizer_bal,
'/content/drive/MyDrive/balanced_split/vectorizer_bal.pkl')
# --- Display shapes ---
print("Shapes:")
print("X_train_vec:", X_train_bal_vec.shape)
print("X_test_vec:", X_test_bal_vec.shape)
print("y_train:", y_train_bal.shape)
print("y_test:", y_test_bal.shape)
```

Output :

```
Shapes:
X_train_vec: (8000, 20000)
X_test_vec: (2000, 20000)
y_train: (8000,)
y_test: (2000,)
```

Percentage of test and training set

Percentage of training and testing has set in the correct ratio.Training set is set to 80% and testing set is set to 20%.

Code :

```
total_samples = len(balanced_df)
```

```
train_size = len(X_train)

test_size = len(X_test)

print("Training set size:", train_size,
f"({train_size/total_samples:.2%})")

print("Test set size:", test_size, f"({test_size/total_samples:.2%})")
```

Output:

```
Training set size: 8000 (80.00%)

Test set size: 2000 (20.00%)
```

11. Model Training

Why is Logistic Regression used?

I chose Logistic Regression as the base algorithm because:

- It is a simple, efficient, and interpretable model for multi-class text classification problems like review rating prediction.
- It performs well with TF-IDF feature representations, which are sparse and high-dimensional.
- It is less prone to overfitting with regularization (L1, L2 penalties).
- It provides a strong baseline for comparison before applying more complex models (e.g., SVM, BERT).

Model Training Logic

Balanced Dataset Model

- Dataset was balanced by each rating class equally (2000 reviews per rating).
- Stratified 80/20 train-test split maintained equal class distribution.
- GridSearchCV was used for hyperparameter tuning with 5-fold cross-validation.
- Parameters tuned:

```
{
   'C': [0.01, 0.1, 1, 10, 100],   # Regularization strength
   'penalty': ['l1', 'l2'],        # Regularization type
   'class_weight': [None, 'balanced']
}
```

- Scoring metric : Accuracy

Evaluation Logic

Metric	Value
Best CV Score	0.4622
Test Accuracy	0.4615
Macro F1 Score	0.45

Interpretation

- Balanced models produced comparable results (≈46% accuracy).
- The balanced dataset ensured fair learning across all rating classes, but reduced data volume may limit diversity.
- The macro F1 score ≈ 0.45 shows moderate predictive power — reasonable for baseline models on noisy text data.

Model Performance

- Both models performed comparably with accuracy around 43–44% and macro F1 around 0.43–0.44.
- Model A (Balanced) achieved slightly better recall, meaning it recognized more true ratings when tested on real-world (imbalanced) data.
- The minor difference in scores shows both models generalize decently but in different ways — Model A is fairer across classes, while Model B is slightly more accurate overall.

Observations on How Training Data Distribution Affects Generalization

- Reduces model bias toward frequently occurring ratings (like 5-star reviews).
- Helps the model learn to treat all classes equally, improving fairness and recall.
- However, it reduces data diversity, as undersampling discards many natural examples from majority classes.

Which Model Would I Deploy and Why

Recommended Model: Model B (Imbalanced + Class Weight Balanced)

Reasons:

1. Better Generalization – Performs slightly better (Accuracy = 0.4365, F1 = 0.4368) when tested on balanced data.

2. Data Retention – Trains on the full dataset, preserving natural review variety instead of discarding samples.

3. Real-World Adaptability – Real review data is rarely balanced; this model mirrors real deployment scenarios.

4. Internal Bias Control – `class_weight='balanced'` compensates for skewed data without needing artificial sampling.

5. Deployment Efficiency – Requires less preprocessing and scales well with new incoming reviews.

While both models are valid baselines, Model B (trained on imbalanced data with balanced class weighting) offers a better trade-off between accuracy, fairness, and scalability, making it the ideal choice for deployment in a real-world Automated Review Rating System.