

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

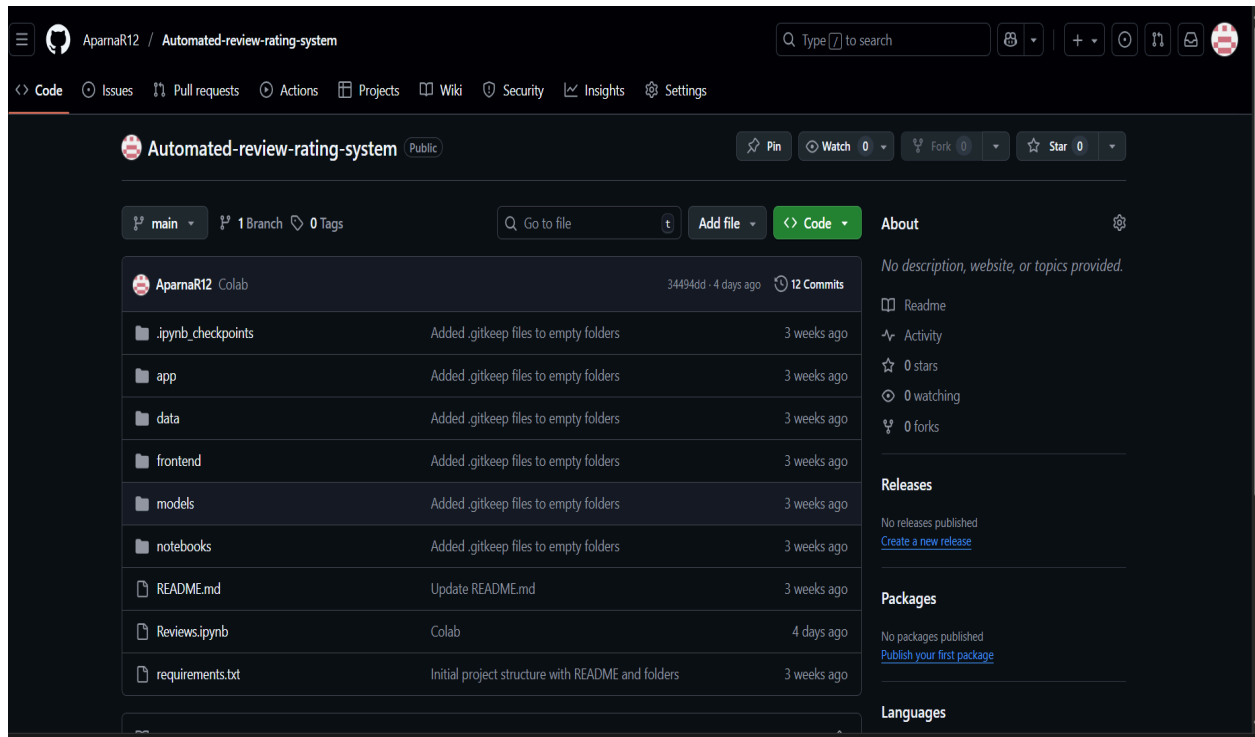
It was developed and executed in **Google Colab**, a cloud-based Jupyter notebook environment that supports GPU acceleration and seamless Python workflows. The implementation was carried out using Python 3.10.

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/dongrelaxman/amazon-reviews-dataset), <https://www.kaggle.com/dongrelaxman/amazon-reviews-dataset>, <https://www.kaggle.com/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 | A18K1ODH112MVB | B007WTAJTO | Omie | [0, 0] | Purchased this for my device, it worked as adv... | 5 | MOAR SPACE!!! | 1382659200 | 10/25/2013 | 409 | 0 | 0 |
| 2 | A2FII3I2MBMUJA | 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 | dli pa | 0 | 0 | 1 | 1346976000 | Not as Advertised | Product arrived labeled as Jumbo Salted Peanut... |
| 2 | 3 | B000LQOCHO | 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 | A1UQRSOLF8GW1T | 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/5ddb429478d88251550610e | 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)
```

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 stopwords removal and lemmatization more consistent.

Code :

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

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)
```

Remove URLs, HTML Tags, Punctuation, and Special Characters etc

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

- URLs (<http://>, www. etc.)
- HTML tags (
, <div>)
- 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 stopwords 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. Imbalanced Dataset

Imbalanced dataset examines the distribution of ratings to understand how skewed the data is. The goal is not to balance the dataset, but rather to intentionally sample reviews in a way that reflects a predefined rating distribution. To achieve this, the code defines specific percentages for each rating category—like 20% for 5.0, 30% for 4.0, 25% for 3.0, 15% for 2.0 and 10% for 1.0. This imbalanced dataset is useful for simulating real-world conditions where certain ratings dominate, and it allows for testing how models perform when trained on skewed data. The final output confirms the shape of the new dataset and prints the rating distribution to verify that the sampling worked as intended.

Code :

```
import pandas as pd

df_remaining = merged_df.merge(
    balanced_df[['Rating', 'Reviews']],
    on=['Rating', 'Reviews'],
    how='left',
    indicator=True
)
```


```

df_remaining = (
    df_remaining[df_remaining['_merge'] == 'left_only']
    .drop(columns=['_merge'])
    .reset_index(drop=True)
)

print("Remaining dataset shape after removing balanced:",
      df_remaining.shape)

# Desired ratios for each rating

ratios = {
    1: 0.10, # 10%
    2: 0.15, # 15%
    3: 0.25, # 25%
    4: 0.30, # 30%
    5: 0.20 # 20%
}

#  Set total size of unbalanced dataset

total_unbalanced = 10000

unbalanced_df_list = []

for rating, ratio in ratios.items():
    df_rating = df_remaining[df_remaining['Rating'] == rating]
    sample_size = int(total_unbalanced * ratio)
    # Handle cases where not enough samples are available
    if len(df_rating) >= sample_size:
        df_sampled = df_rating.sample(sample_size, random_state=42)
    else:
        print(f" Only {len(df_rating)} samples available for rating {rating}, expected {sample_size}. Taking all available.")
        df_sampled = df_rating

```

```

unbalanced_df_list.append(df_sampled)

df_unbalanced = pd.concat(unbalanced_df_list).reset_index(drop=True)

# Shuffle to mix the samples well

df_unbalanced = df_unbalanced.sample(frac=1,
random_state=42).reset_index(drop=True)

print("✅ Final unbalanced dataset shape:", df_unbalanced.shape)

print("\n Rating distribution (%):")

print(df_unbalanced['Rating'].value_counts(normalize=True) * 100)

```

Output :

Remaining dataset shape after removing balanced: (62281, 3)

✅ Final unbalanced dataset shape: (10000, 3)

Rating distribution (%):

Rating

4 30.0

3 25.0

5 20.0

2 15.0

1 10.0

Name: proportion, dtype: float64

8. Visualization

Bar Chart

Bar Chart visually represents how ratings are distributed in the dataset. It shows how the counts of review are there in each rating(1-5 star).

Code :

```

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

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

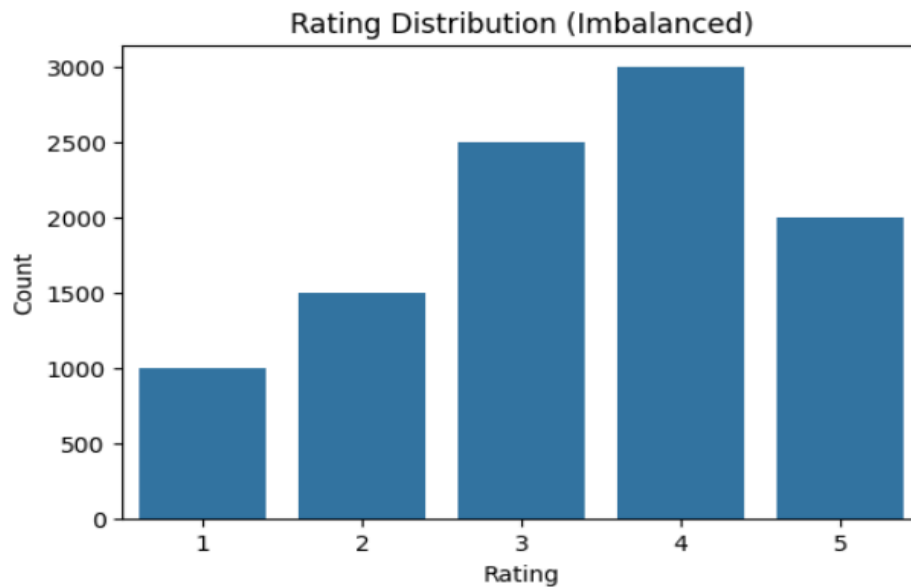
```

```
plt.title("Rating Distribution (Imbalanced)")

plt.xlabel("Rating")

plt.ylabel("Count")

plt.show()
```



Histogram

The histogram provides a visual summary of how many words are frequently used in the reviews.

Code :

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

plt.hist(df_unbalanced["review_length"], bins=50)

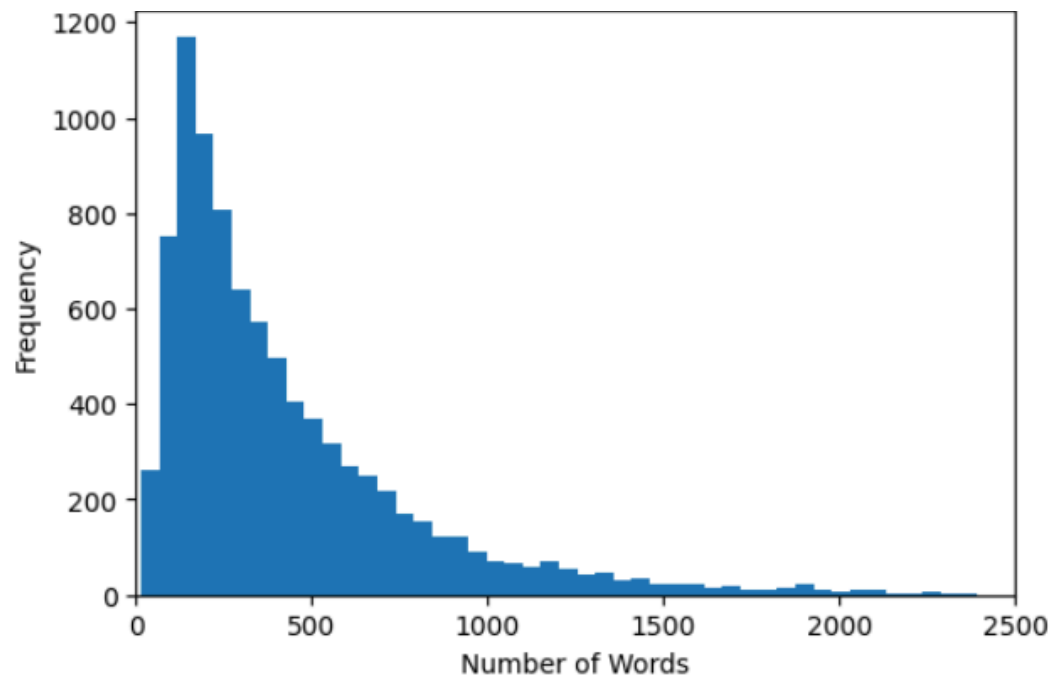
plt.title("Review Length Distribution")

plt.xlabel("Number of Words")

plt.xlim(0, 4500)

plt.ylabel("Frequency")

plt.show
```



Box Plot

The boxplot visualizes how the length of reviews varies across different rating scores. Each box represents the distribution of review lengths for a specific rating—from 1.0 to 5.0.

Code :

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

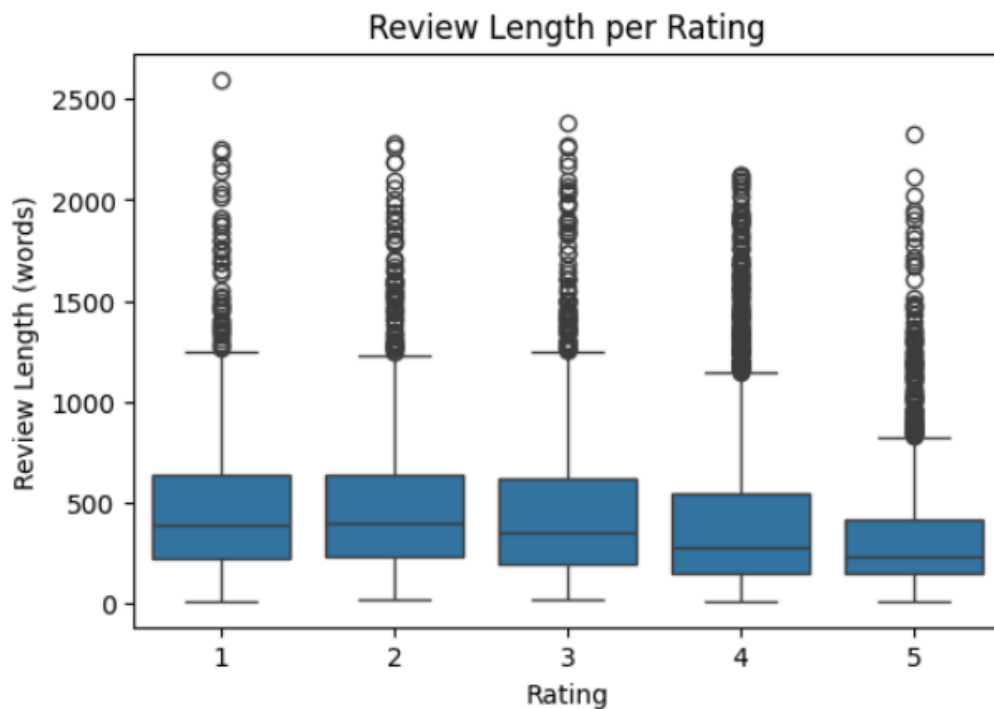
sns.boxplot(x="Rating", y="review_length", data=df_unbalanced)

plt.title("Review Length per Rating")

plt.xlabel("Rating")

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

plt.show()
```



Violin Plot

The violin plot provides a rich visualization of how review lengths vary across different rating scores. Each violin shape represents the distribution of word counts for a specific rating, combining the features of a boxplot with a kernel density estimate.

Code :

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

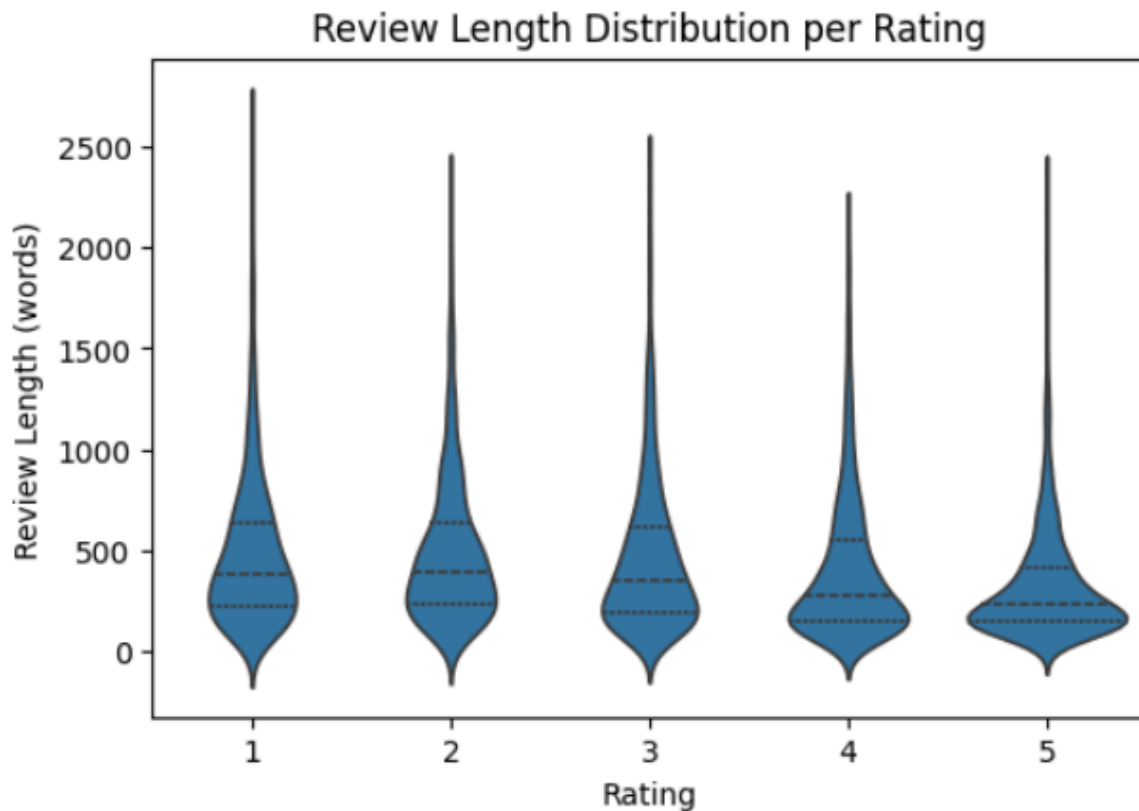
sns.violinplot(x="Rating", y="review_length", data=df_unbalanced,
inner="quartile")

plt.title("Review Length Distribution per Rating")

plt.xlabel("Rating")

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

plt.show()
```



Display 5 sample reviews per rating

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

Code :

```
for rating in sorted(df_unbalanced["Rating"].unique()):  
    print(f"\n--- Rating {rating} ---")  
    samples = df_unbalanced[df_unbalanced["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 bad thing ever encounter blocked order provide bunch document screenshot bank app show card number bunch card account statement copy passport even twice completely blocked account enough understand actually make payment card recruit moron employee amazon account closed the information provide insufficient verify payment ownership pending order cancel believe error please contact amazon customer service
2. enjoy syrup hate constant increase get reeeally feed corporate greed country
3. person india need know treat customer instead go superior amazon fault agent priyanka bad ego fit inside chatbox learn c apply cs job
4. hop find delicious new low carb snack well miss mark delicious mile extremely salty chewy hop find way salvage anyone suggestion ear
5. disgust customer care rude unhelpful unprofessional keep refuse answer question chat feel would look bad end chat end still type awful people

--- Rating 2 ---

1. gum stick teeth flavor last long ok vending machine gum decent price gum good
2. amazon one point amaze understand delay ship crazy time ship time keep get longer longer wish membership cost reflect inconvenience since two day delivery nonexistent
3. sure happen amazon feel like care mess anymore actually seem like mess often get right delivery come say drop package neighbor place customer service absolute joke say dealt kind customer service rep last night one leave hold min hang without ever come back call back dealt rep call parag actually help rude way like usual get think every time call amazon treat like something wrong hello amazon people complain complain someone patience deal automated system interrogation order speak someone must really bad experience know care delivery get postponed next day send message sorry order delay cancel order get full refund bring pay well amazon get bad price like use
4. like large variety oatmeal less week purchase amazon amazon prime march th go costco find exact product save money buy somewhere else
5. find watery least can pumpkin likely less watery fresh pumpkin puree definitely watery libby generic brand grocery store texture upon open organic one smooth irregular make pie taste bit bland well also make mistake crust much water hand pie disappear two day rather one really

liked art though label nice basically make fantastic pie especially would like organic suggest strain coffee filter place mesh colander overnight longer

--- Rating 3 ---

1. melitta cafe riviera sunset coffee bad sadly great either something would buy mainly probably forget

2. general walden farm dressing taste great make diet less monotonous husband like always prefers high calorie version anything ranch bland bad taste great possibly could add little garlic onion powder make dip

3. dark chocolate raspberry two favorite flavor together however difficult get dark chocolate enough cacao give health benefit need justify eat chocolate need high protein bar taste good fit bill

4. grown sugar maple country buy syrup farmer unmarked gallon can store syrup never enter house plenty experience maple syrup tip maple tree maple syrup organic definition farmer tap tree add nothing pay organic house sugar every winter know talk read idea may happy product good syrup price right purchase good buy shy purchasing really star minus star good grade b syrup normally maple caramel note syrup lack strong maple flavor make suitable cooking good match like strong flavor complexity finesse grade amber short rather bland pedestrian maple syrup know whether tree tap label vary year year batch uninteresting buy alternative whole food label quite frankly whole food grade amber strong maple flavor complexity whole food grade b significantly strong flavor would rate whole food grade amber star minus grade b star

5. best gluten free ritz like cracker exactly praise cheese bad plain one taste like egg lot gluten free bread cracker use egg lack glue always taste yuck say plain one ok make peanut butter sandwich cracker pretty good dip chicken tuna salad sigh

--- Rating 4 ---

1. open package trepidation base review rinse fishy smell liquid drop noodle simmer tomato sauce wait minute plat pigged texture odd unpleasant noodle absorb sauce flavor felt satisfied full ask

2. serious coffee drinker grid bean brew coffee yes taste difference burr grinder blade grinder oh yeah serious probably imagine instant coffee year because normally taste awful sour chemical taste fragrance friend tell via giggle call one joked use coffee flavor milk appologize one day another friend ask buy instant coffee laughed request friend much buy small via package possible tasted surprise sour taste chemical taste low ignore especially drink coffee sugar strongly like skinny long bag choose scissors teeth open bag bag tear open easily top finger course alert hide away toddler next thing brainer dump powder coffee mug pour hot water

doubt really need stir anyway smell strong fresh burr grind coffee good enough believe coffee confess almost helded breath first sip ready sort awful taste surprise taste sour trouble drink whole cup lick lip think hmm bad second think hot water away another cup come final judgement instant coffee awesome grinder loud noise carry anywhere go scissors need good cup coffee hot water away replace old fashion fresh grind coffee still yet keep around the house time house need dead quiet crave cup coffee star still good fresh grind coffee

3. go get excited soup mile away soup pot acceptable alternative sprinkle fresh bacon crumbles mine since can version flavor lack texture

4. good digestive crunchy sweet reasonable especially subscription great cup tea

5. start eat almond healthy alternative fatty high carb snack want something satisfy sweet tooth occasion cocoa roast almonds fit bill nicely dust cocoa fat unhealthy stuff would find regular almond still get nice chocolate taste little bit sweeter really want like dark chocolate semisweet bittersweet still great awhile snack

--- Rating 5 ---

1. order christmas gift arrive time destination tell taste good

2. great long lasting chew small dog yorkie spend happy hour two new chew return later gnaw chew put samson small dog need big name tranquil brain neutral state wife feel good poisoning make china junk besides chew stink low quality brand try

3. find single place price love little glass pitcher single coffee maker

4. perfect need huge fan loose leaf tea mild white tea deep rich strong black tea realize thing miss inventory jasmine tea almost identical one serve fine chinese eatery voila sunflower jasmine fit bill hop nice smooth easy drink tea go well meal pleased

5. wellness brand one vet recommend overweight cat say brand filler can cat food since use three cat overweight cat lose weight regain vim vigor arthritis well three cat love stuff

-10. Train Test Split

To split the dataset into training and testing sets while preserving class balance, clean the text data, and convert it into numerical features using TF-IDF vectorization.

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.

Train-Test Split with Stratification

The dataset is split into training and testing sets using `train_test_split`, with `stratify=y` to ensure that the distribution of ratings (classes) remains consistent across both sets. This is crucial when working with imbalanced datasets, as it prevents the model from being trained or evaluated on skewed subsets.

Independent Text Preprocessing

The `clean_text` function is applied separately to `X_train` and `X_test`. This ensures that preprocessing (like removing stopwords, punctuation, etc.) is done independently for each set, avoiding data leakage from test to train. Any resulting NaN values are dropped to maintain data integrity.

Reset Indices

Both `X` and `y` for train and test sets are reset to have clean, sequential indices. This simplifies downstream processing and avoids index mismatches.

TF-IDF Vectorization

The cleaned text is transformed into numerical features using `TfidfVectorizer`, which converts each review into a vector of TF-IDF scores. The `max_features=5000` parameter limits the vocabulary size to the top 5000 terms, balancing expressiveness with computational efficiency.

TF = Term Frequency in document / Total Words in Document

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

Code :

```
import pandas as pd

import joblib

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer
```

```

# --- Step 1: Define desired ratios and total size ---

ratios = {
    1: 0.10, # 10%
    2: 0.15, # 15%
    3: 0.25, # 25%
    4: 0.30, # 30%
    5: 0.20 # 20%
}

total_samples = 10000

unbalanced_df_list = []

# --- Step 2: Sample according to ratios ---

for rating, ratio in ratios.items():
    df_rating = df_unbalanced[df_unbalanced['Rating'] == rating]
    sample_size = int(total_samples * ratio)
    if len(df_rating) >= sample_size:
        df_sampled = df_rating.sample(sample_size, random_state=42)
    else:
        # If fewer rows available, take all
        df_sampled = df_rating

        print(f"⚠ Rating {rating} has only {len(df_rating)} samples,
taking all available.")

    unbalanced_df_list.append(df_sampled)

# ✅ Final dataset name remains df_unbalanced

df_unbalanced = pd.concat(unbalanced_df_list).reset_index(drop=True)

print("✅ Sampled rating distribution:")

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

print("Total samples:", len(df_unbalanced))

# --- Step 3: Train-Test Split (stratified) ---

```

```

X_unbal = df_unbalanced["Reviews"]

y_unbal = df_unbalanced["Rating"]

X_train_unbal_raw, X_test_unbal_raw, y_train_unbal_raw, y_test_unbal_raw =
train_test_split(

    X_unbal, y_unbal,

    stratify=y_unbal,

    test_size=0.2,

    random_state=42,

    shuffle=True
)

# --- Step 4: Text cleaning ---

X_train_unbal = X_train_unbal_raw.apply(clean_review)
X_test_unbal = X_test_unbal_raw.apply(clean_review)

# Keep only valid (non-empty) rows

train_valid = X_train_unbal.notna()
test_valid = X_test_unbal.notna()

X_train_unbal = X_train_unbal[train_valid].reset_index(drop=True)
y_train_unbal = y_train_unbal_raw[train_valid].reset_index(drop=True)
X_test_unbal = X_test_unbal[test_valid].reset_index(drop=True)
y_test_unbal = y_test_unbal_raw[test_valid].reset_index(drop=True)

# --- Step 5: TF-IDF Vectorization ---

vectorizer_unbal = TfidfVectorizer(

    max_features=20000,

    ngram_range=(1, 3),

    sublinear_tf=True,

    stop_words='english'
)

X_train_unbal_vec = vectorizer_unbal.fit_transform(X_train_unbal)

```

```

X_test_unbal_vec = vectorizer_unbal.transform(X_test_unbal)

# --- Step 6: Save vectorizer for reuse ---

joblib.dump(vectorizer_unbal,
'/content/drive/MyDrive/unbalanced_split/vectorizer_unbal.pkl')

# --- Step 7: Final output ---

print("\n✅ Final Unbalanced Split Shapes:")

print("X_train_unbal_vec:", X_train_unbal_vec.shape)

print("X_test_unbal_vec:", X_test_unbal_vec.shape)

print("y_train_unbal:", y_train_unbal.shape)

print("y_test_unbal:", y_test_unbal.shape)

```

Output :

✅ Sampled rating distribution:

Rating

| | |
|---|------|
| 4 | 3000 |
| 3 | 2500 |
| 5 | 2000 |
| 2 | 1500 |
| 1 | 1000 |

Name: count, dtype: int64

Total samples: 10000

✅ Final Unbalanced Split Shapes:

X_train_unbal_vec: (8000, 20000)

X_test_unbal_vec: (2000, 20000)

y_train_unbal: (8000,)

y_test_unbal: (2000,)

Percentage of testing and training set

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

Code :

```
total_samples = len(df_unbalanced)

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 is 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

Imbalanced Dataset Model

- Used the original dataset without balancing.
- `class_weight='balanced'` was applied to handle imbalance internally.
- Tuned parameters:


```
{  
  'C': [0.01, 0.1, 1, 10]  
  'solver': ['liblinear', 'lbfgs']  
}
```

- Scoring metric : Accuracy

Evaluation Logic

| Metric | Value |
|----------------|-------|
| Best CV Score | 0.442 |
| Test Accuracy | 0.456 |
| Macro F1 Score | 0.46 |

Interpretation

- Imbalanced models produced comparable results ($\approx 46\%$ accuracy).
- The imbalanced dataset preserved data richness but required class weighting to mitigate bias.
- The macro F1 score ≈ 0.45 – 0.46 shows moderate predictive power — reasonable for baseline models on noisy text data.
- ROC-AUC of 0.77 (imbalanced model) shows good discriminative ability even with skewed data.

Model Performance

- Both models performed comparably with accuracy around 43–44% and macro F1 around 0.43–0.44.
- Model B (Imbalanced) showed better precision and F1 score, indicating stronger prediction confidence and consistency across balanced test 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

- Preserves natural data distribution, giving the model exposure to realistic patterns.
- The model tends to favor majority classes slightly but learns richer linguistic variations.
- Using `class_weight='balanced'` mitigates most bias without data loss.
- Leads to slightly better generalization since the model learns from a broader set of examples.

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.