AUTOMATED REVIEW SYSTEM

1. Project Overview

An intelligent system that analyzes customer reviews and predicts corresponding star ratings. It uses natural language processing (NLP) to extract sentiment and key features from textual feedback. The model is trained on labeled review data to learn patterns between language and rating.

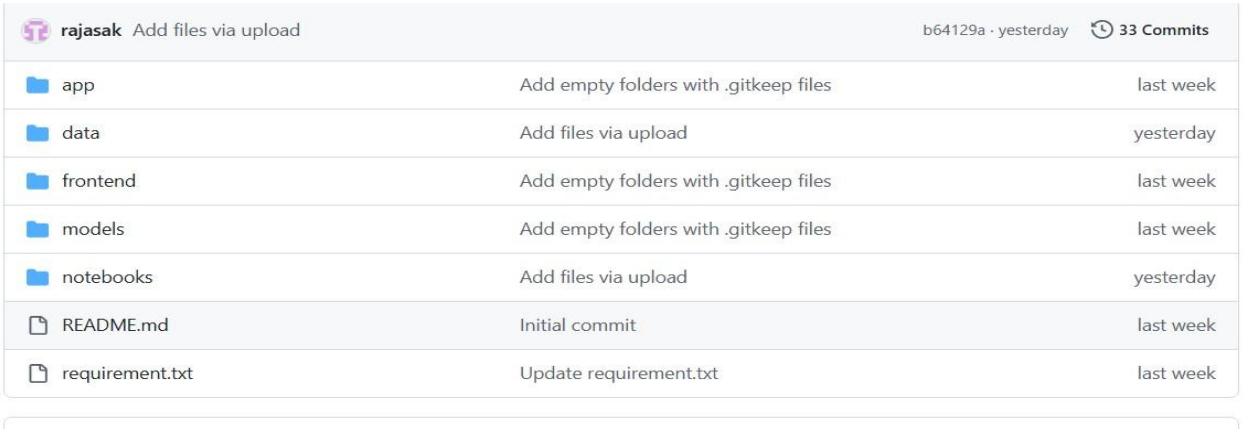
2. Environment Setup

- Installed Python 3.10+ with required libraries: pandas, numpy, matplotlib, seaborn, spacy, scikit-learn, beautifulsoup4, requests, Torch, Transformers
- IDE used: VS Code
- Verified proper functioning of packages for data preprocessing, NLP, and visualization

3. GitHub Project Setup

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

Structure of directory



4. Data Collection

- Dataset collected has 5 lakh rows
- Dataset was related to food product
- link=https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews
- Summary column was concatenated with review column
- Final dataset contains 2 column with Review and Rating
- 1 star=29893, 2 star = 16234, 3 star= 24260, 4 star=42518 , 5 star= 198577

4.1. Final dataset

- Link for final dataset= https://drive.google.com/file/d/1PEbxA8R9GeC7tRiIjFuos4IWZJLsj9Q_/view?usp=sharing
- Final dataset contain 1.6 lakh rows
- Final dataset was created 1 star=29893, 2 star=16234, 3 star=24260, 4 star=42517, 5 star=50000

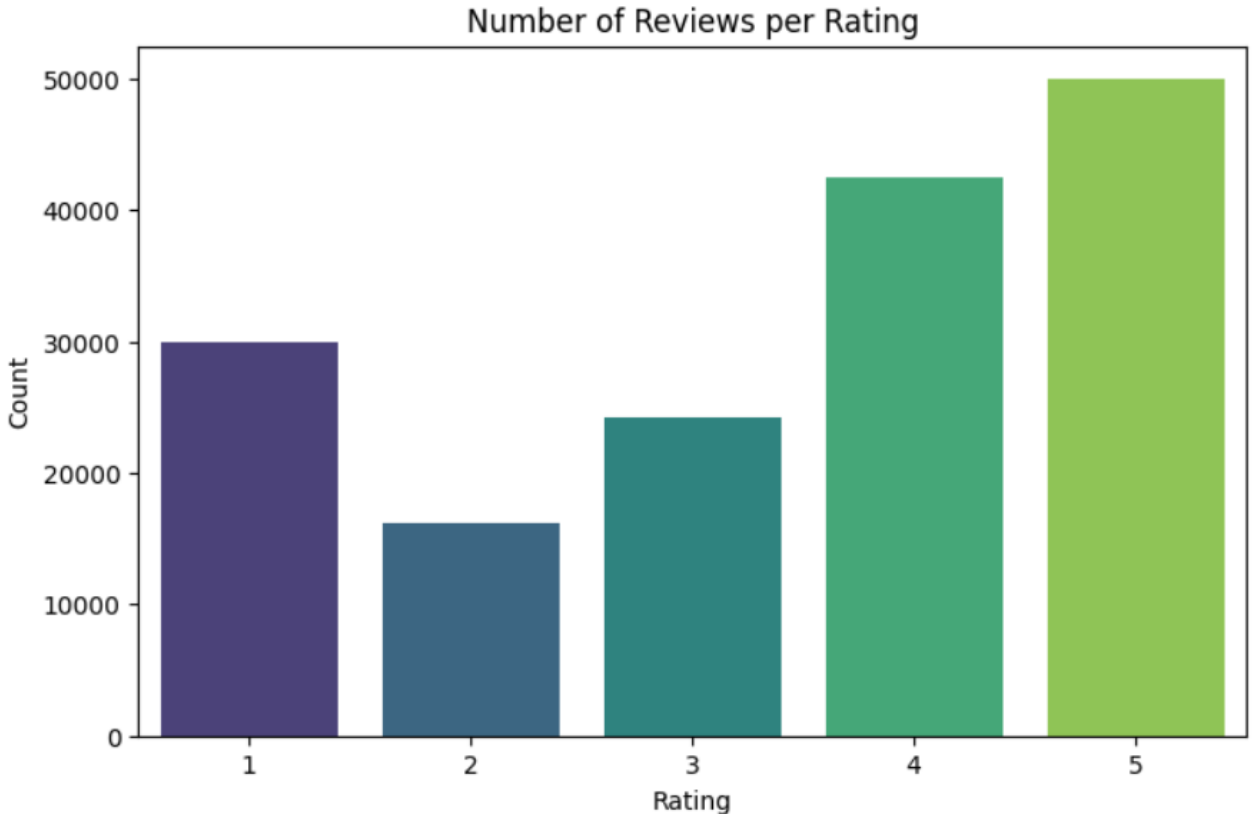


Fig count plot

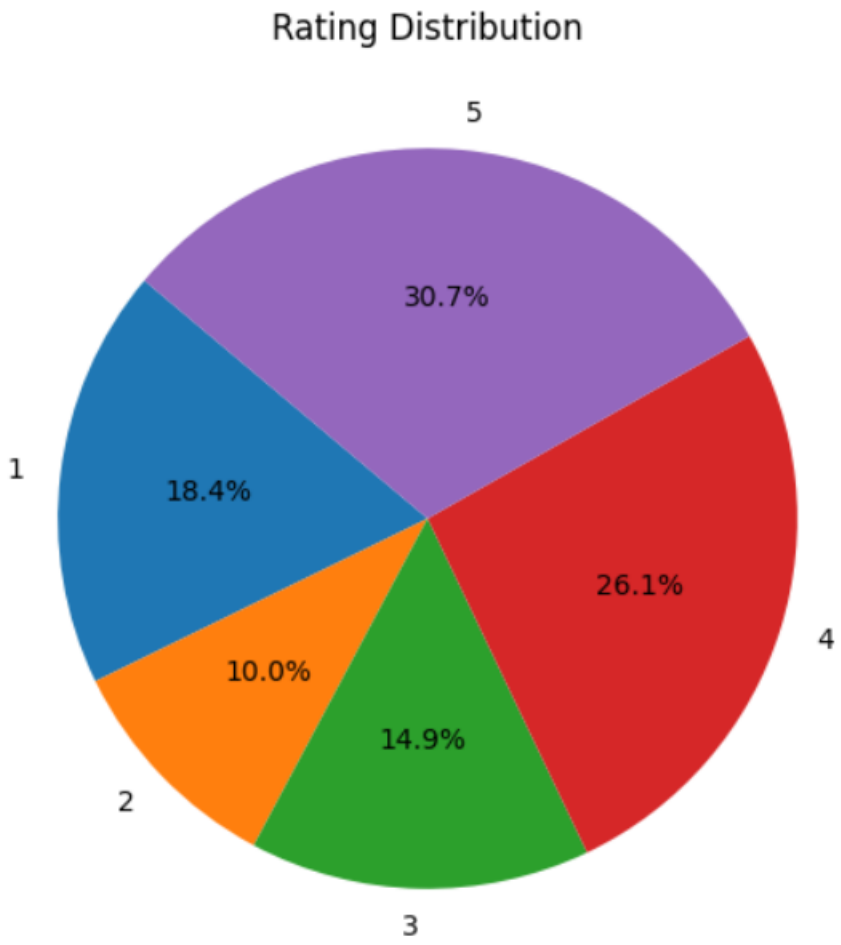


Fig pie chart

5. Data Preprocessing

Effective data preprocessing is essential to improve the performance and accuracy of machine learning models. The following techniques were applied to prepare the raw review dataset for modeling.

5.1 Removing Duplicates

Duplicate rows containing the exact same review and rating were removed to prevent bias and overfitting. This ensured that the dataset only included unique observations.

Code:

```
has_duplicates = df.duplicated().any()
print(has_duplicates)
num_duplicates =df.duplicated().sum()
print(f"Number of duplicate rows: {num_duplicates}")
df.drop_duplicates(inplace=True)
```

5.2 Removing Conflicting Reviews

Some reviews had identical text but different star ratings. These inconsistencies can confuse the model. Such conflicting entries were identified and removed to maintain label clarity.

code:

```
# Filter to only reviews that have more than one unique rating
conflicting_reviews = conflicting_counts[conflicting_counts > 1]
print("Number of conflicting review texts:", conflicting_reviews.shape[0])


conflicting_reviews = df.groupby('Review')['Rating'].nunique()
conflicting_reviews = conflicting_reviews[conflicting_reviews > 1].index
df = df[df['Review'].isin(conflicting_reviews) == False]
```

5.3 Handling Missing Values

Rows with missing or null values—particularly in the review text or rating column—were removed. This step ensured the dataset was complete and meaningful for analysis.

Code:

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

5.4 Dropping Unnecessary Columns

Non-essential columns such as UserId, ProfileName or user identifiers were dropped. These fields did not contribute to the model and could introduce noise or privacy concerns.

5.5 Lowercasing Text

All review text was converted to lowercase to maintain uniformity. This helps prevent duplication of tokens like "Good" and "good" being treated as separate words.

5.6 Removing URLs

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

5.7 Removing Emojis and Special Characters

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

5.8 Removing Punctuation

Punctuation marks and numbers can disrupt the natural language patterns of the text. By removing them, we simplify the tokenization process and prevent punctuation from being misinterpreted as separate tokens.

Code:

```
df_final['Review'] = df_final['Review'].str.lower()

df_final['Review'] = df_final['Review'].str.replace(r'\[.*?\]', '', regex=True)


def clean_text(text):
    text = str(text)
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)  # remove URLs
    text = re.sub(r'<.*?>', '', text)  # remove HTML tags
    text = re.sub(r'[\U00010000-\U0010ffff]', '', text)  # remove emojis
    text = re.sub(r'[^\w\s]', '', text)  # remove punctuation
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)  # remove special characters
    text = re.sub(r'\s+', ' ', text).strip()  # remove extra whitespace
    return text
```

5.9 Removing non-english reviews

There were total 89 non English reviews in the dataset, those reviews were removed using the library langdetect

Code:

```
def detect_language(text):
    try:
        return detect(str(text))
    except LangDetectException:
        return 'unknown'
# Apply language detection
```

df_final['language'] = df_final['Review'].apply(detect_language)

 # Count non-English reviews

non_english_count = df_final[df_final['language'] != 'en'].shape[0]

5.10 Stopwords Removal

Stopwords, such as "the", "is", and "and", are frequently occurring words that might not add significant semantic value. We use libraries like SpaCy to filter these words out, reducing noise and focusing on words that contribute meaning to the reviews, there were total 326 stop words in spacy.

```
["'d", "'ll", "'m", "'re", "'s", "'ve", 'a', 'about', 'above', 'across', 'after', 'afterwards', '
again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also', 'although', 'always', 'a
m', 'among', 'amongst', 'amount', 'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything',
'anyway', 'anywhere', 'are', 'around', 'as', 'at', 'back', 'be', 'became', 'because', 'become', '
becomes', 'becoming', 'been', 'before', 'beforehand', 'behind', 'being', 'below', 'beside', 'besi
des', 'between', 'beyond', 'both', 'bottom', 'but', 'by', 'ca', 'call', 'can', 'cannot', 'could',
'did', 'do', 'does', 'doing', 'done', 'down', 'due', 'during', 'each', 'eight', 'either', 'eleven
', 'else', 'elsewhere', 'empty', 'enough', 'even', 'ever', 'every', 'everyone', 'everything', 'ev
erywhere', 'except', 'few', 'fifteen', 'fifty', 'first', 'five', 'for', 'former', 'formerly', 'fo
rty', 'four', 'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'have', 'he'
, 'hence', 'her', 'here', 'hereafter', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him',
'himself', 'his', 'how', 'however', 'hundred', 'i', 'if', 'in', 'indeed', 'into', 'is', 'it', 'it
s', 'itself', 'just', 'keep', 'last', 'latter', 'latterly', 'least', 'less', 'made', 'make', 'man
y', 'may', 'me', 'meanwhile', 'might', 'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'muc
h', 'must', 'my', 'myself', "n't", 'name', 'namely', 'neither', 'never', 'nevertheless', 'next',
'nine', 'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere', 'n‘t', 'n’t',
'of', 'off', 'often', 'on', 'once', 'one', 'only', 'onto', 'or', 'other', 'others', 'otherwise',
'our', 'ours', 'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps', 'please', 'put', 'qui
te', 'rather', 're', 'really', 'regarding', 'same', 'say', 'see', 'seem', 'seemed', 'seeming', 's
eems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'six', 'sixty', 'so', 'som
e', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'ta
ke', 'ten', 'than', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'the
reafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'third', 'this', 'thos
e', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top', '
toward', 'towards', 'twelve', 'twenty', 'two', 'under', 'unless', 'until', 'up', 'upon', 'us', 'u
sed', 'using', 'various', 'very', 'via', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when',
'whence', 'whenever', 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'where
ver', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why',
'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yoursel
ves', '‘d', '‘ll', '‘m', '‘re', '‘s', '‘ve', '’d', '’ll', '’m', '’re', '’s', '’ve']
```

Code:

```
#lemmatization and stop word removal
def cleaning(text):
doc = nlp(text)
    return ' '.join([token.lemma_ for token in doc if not token.is_stop])
```

5.11 Lemmatization

Lemmatization is a text preprocessing technique that reduces words to their base or dictionary form, known as the **lemma**. For example, words like **"running"**, **"ran"**, and **"runs"** are all reduced to the base form **"run"**. Unlike stemming, lemmatization uses linguistic rules and vocabulary, ensuring that the resulting word is meaningful. This helps in grouping similar words and improves model performance by reducing the size of the vocabulary without losing context.

Why lemmatization is better than stemming

Lemmatization is often preferred over stemming because:

1. **Produces Real Words**:

   Lemmatization returns valid dictionary words (e.g., *"running"* → *"run"*), whereas stemming may return incomplete or non-existent words (e.g., *"running"* → *"runn"*).

2. **Context-Aware**:

   Lemmatization considers the context and part of speech (POS) of a word to find its correct root form.
   Stemming blindly trims word endings without understanding grammar.

3. **More Accurate and Clean**:

   Lemmatization provides cleaner and more accurate tokens, which improves model understanding and performance, especially in NLP tasks like sentiment analysis or classification.

5.12 Filtering by Word Count

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

- Reviews with fewer than 3 words.
- Reviews that exceed 200 words.
  This ensures that the dataset remains robust and relevant for model training.

Code:

```
df_final = df_final[(df_final['word_count'] >= 3) & (df_final['word_count'] <= 200)]
```

5.13 Label Encoding for Classification

Since BERT's classification head expects class indices starting from 0, the original star ratings (ranging from 1 to 5) were mapped to class labels 0 to 4.

```
df['label'] = df['Rating'].astype(int) – 1
```

6. Data visualization

Box plot

A **box plot** is a visual tool used to display the distribution of numerical data, showing the median, quartiles, and potential outliers. It helps quickly understand how data is spread and identify any extreme values. In this project, box plots were used to compare the word count distribution of reviews across different rating classes.

Before Cleaning:

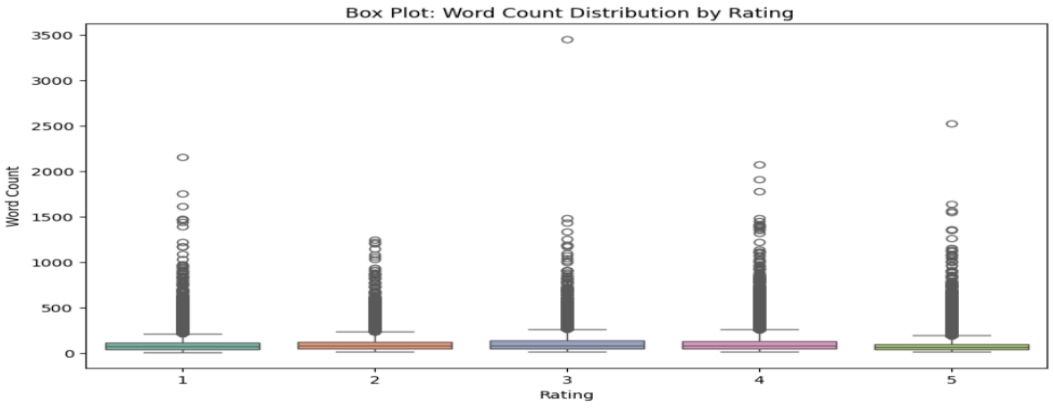


Fig box plot (before cleaning)

After cleaning:

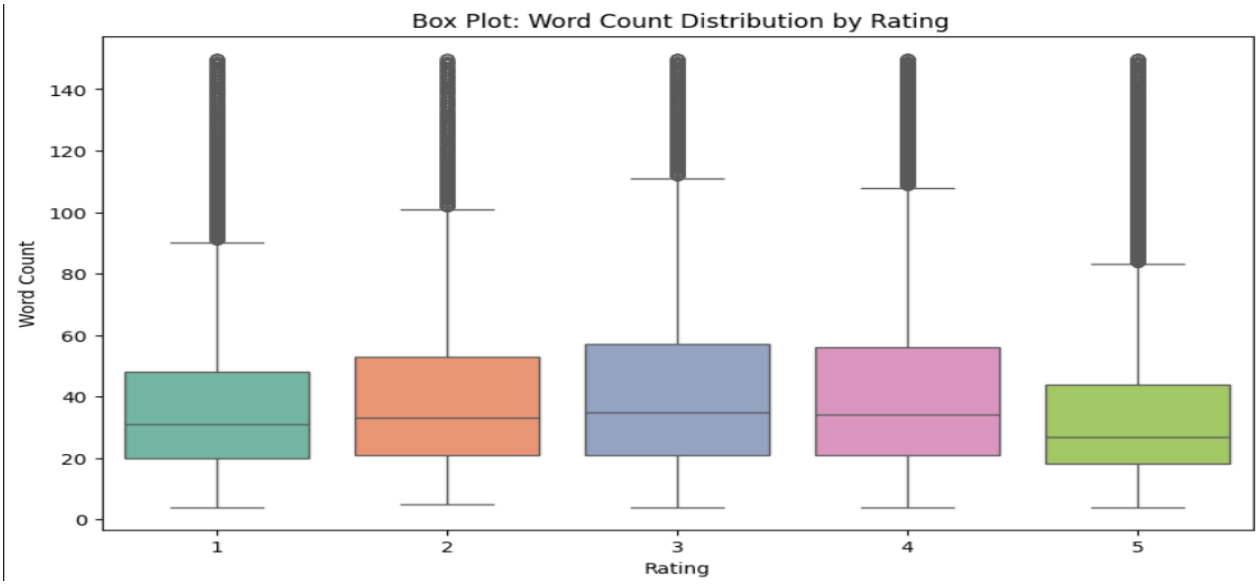


Fig box plot (After cleaning)

Code:

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Rating', y='word_count', data=df, palette='Set2')
plt.title(" Box Plot: Word Count Distribution by Rating")
plt.xlabel("Rating")
plt.ylabel("Word Count")
plt.show()
```

Histogram

A **histogram** is a graphical representation that shows the distribution of a numeric variable by dividing the data into intervals (bins) and counting how many values fall into each bin. In this project, histograms were used to visualize the **word count distribution** of reviews, helping to identify common review lengths and spot patterns across different rating levels.

Code:

```
plt.figure(figsize=(12, 6))
sns.histplot(data=df, x='word_count', hue='Rating', multiple='stack', palette='Set2', bins=30)
plt.title(" Histogram: Word Count Distribution by Rating")
plt.xlabel("Word Count")
plt.ylabel("Number of Reviews")
plt.legend(title='Rating')
plt.show()
```

Before cleaning:

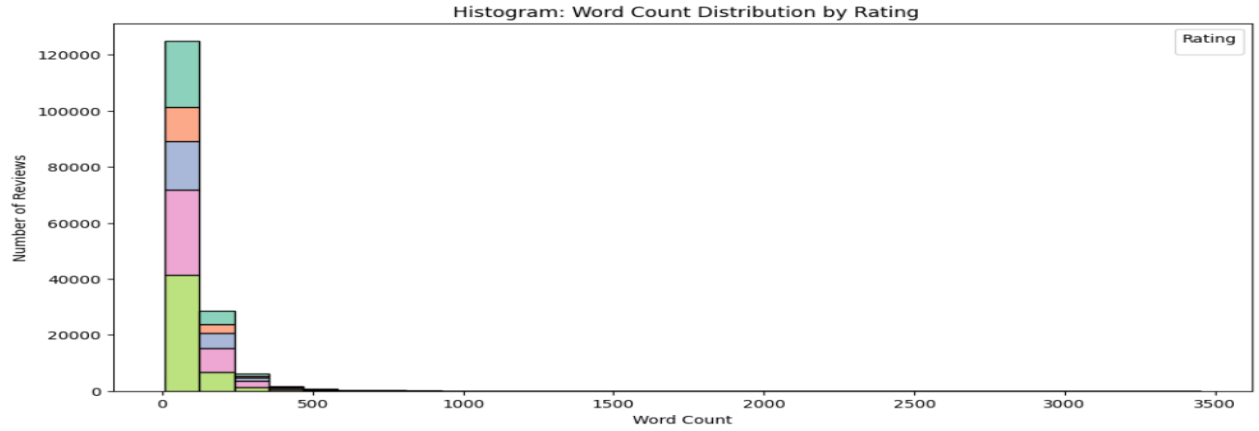


Fig histogram (Before cleaning)

After cleaning:

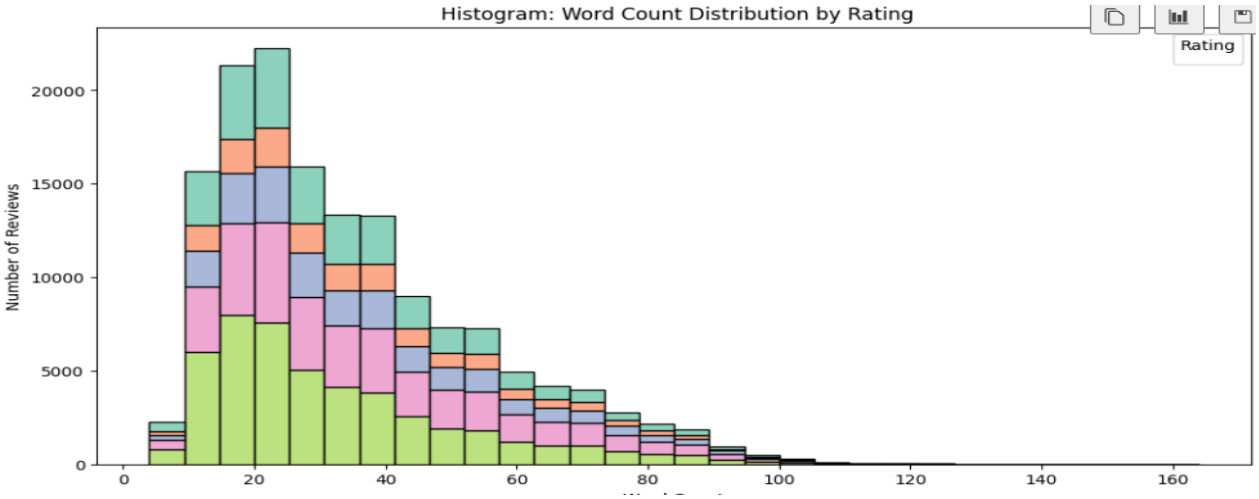


Fig histogram (After cleaning)

Minimum word count in each rating (Before cleaning)

```
 Minimum word count per Rating:
Rating
1     9
2    13
3    12
4    11
5    11
```

Minimum word count in each rating (After cleaning)

```
 Min word count per Rating:
Rating
1    5
2    9
3    9
4    3
5    2
```

```
Minimum word count for Rating 5: 2
Reviews with minimum word count:
                Review  word_count
90323  producte review           2
```

Then word count less than 3 were removed

Minimum word count in each rating (After stop word removal and lemmatization)

```
 Min word count per Rating:
Rating
1    4
2    5
3    4
4    2
5    4
```

```
 Minimum word count for Rating 4: 2
 Reviews with minimum word count:
              Review  word_count
94687  work promise           2
```

Then word count less than 3 were removed

Minimum word count in each rating (final dataset)

```
 Minimum word count per Rating:
Rating
1    4
2    5
3    4
4    4
5    4
```

Output

```
  Minimum word count for Rating 1: 4
  Reviews with minimum word count:
                        Review  word_count
 15100  bread stick sorry like           4
 27900   happen length 130 min           4
```

```
 Minimum word count for Rating 2: 5
 Reviews with minimum word count:
                          Review  word_count
146190  not care dry flavor return          5
```

```
 Minimum word count for Rating 3: 4
 Reviews with minimum word count:
                            Review  word_count
36626  bam expect interesting product          4
```

```
 Minimum word count for Rating 4: 4
 Reviews with minimum word count:
                       Review  word_count
159391  lay review enjoy taste           4
```

```
 Minimum word count for Rating 5: 4
 Reviews with minimum word count:
                             Review  word_count
107169  title product advertise seven          4
123767           good movie good copy          4
```

Code:

```
min_word_count_per_rating = df.groupby('Rating')['word_count'].min()

# Display the result

print(" Min word count per Rating:")

print(min_word_count_per_rating)

df_final = df_final[(df_final['word_count'] >= 3) & (df_final['word_count'] <= 200)]
```

Maximum word count in each rating (before cleaning)

```
 Max word count per Rating:
Rating
1    2157
2    1241
3    3451
4    2070
5    2526
```

Maximum word count in each rating (After cleaning)

```
 Max word count per Rating:
Rating
1    2045
2    1205
3    3389
4    1994
5    2391
```

Then word count higher than 200 were removed

Maximum word count in each rating (After stop word removal and lemmatization)

```
 Max word count per Rating:
Rating
1    142
2    130
3    142
4    164
5    131
```

Code:

```
max_word_count_per_rating = df.groupby('Rating')['word_count'].max()


# Display the result
print(" Max word count per Rating:")
print(max_word_count_per_rating)
```

Samples of 1 star rating:

Showing 5 sample reviews for Rating 1:

1. 15 year later mislead review write april 2010 person mention follow title mislead understand 9 pound coffee sampler case think 15 year multiple bad review amazon clarify issue purchase expensive expect not care couple silly mug box opinion know trick people not want change disappointed

2. yuck contain aspartame alternative list shame want like fiber texture great nice crunchythough make think little pet food xd fiber content super aspartame dealbreaker consume aspartame chew gum 14 can diet coke day discover actually give migraine hike anxiety level not migraine panic attack 4 year quit accidentally headache start feel nervous right away study brain tumor controversial avoid like plaguebut find today wonderful alternative fiber smart bran whopping 13 gram fiber taste pretty good nasty artificial sweetener not contain high fructose corn syrup lovely general mill change fiber healthy 1020 calorie cereal not worth headache health impact aspartameplus nice cereal decent aftertaste

3. inedible sorry true read review product vow buy pick trip market remember nasty one think chewy chocolate gfcf great give year old excite look like get hunk chocolate sniff refuse eat try little flavor texture unpleasant hard bite not imagine company thinkingit shame hard find delicious allergyfriendly snack product offensive booooo

4. wow amazon 2200 vs 985 wallmart joke 1300 difference price amazon rip offlol mean formula online time take amazon corp office research think people not care

5. dill big fan simply organic purchase product regularlyhowever particular dill weed fall short expectationsthe flavor lackluster unimpressive order different company find flavorwhat huge disappointment

Samples of 2 star rating

Showing 5 sample reviews for Rating 2:

1. extremely spicy know indian food spicy not list ingredient think d shot big piece green not think human eat dig big green thing dilute lot dairy tone rice potato eat night indigestionit probably ok like spicy food

2. old tough likely order product old past pull date 2 month

3. funky taste disappointed product horrible definitely overpriced well get 12 cent packet sugar free drink mix target add tequila

4. not beat price gloria jean cheapo brand keurig machine show ve get honesty regular 24pod box subtle taste hazelnut boy subtle flavor weak coffee not strong low 6 oz set forget 8 oz level reuse podswhen get keurig get selection green mountain coffee think firstclass kind pricey gloria jean 36 pod save fair 68 pod day difference quality not think worth worry coffee fix door couple box gloria jean case run good stuff not use regularly ll find give keurig machine

5. not smell like coffee coffee not taste bad not smell like coffee non dairy creamer bad smell stop drink look good strong coffee enjoy

Samples of 3 star rating

Showing 5 sample reviews for Rating 3:

1. s use xylosweet xylitol talk xylosweet diarhhea mean 2 teaspoon vs 1 corn product sooxylosweet sweet sugar need little bit guess happen not actually bake like cup cause ve scared maybe bake elimate effect ill let try firstnow food make xylitol mix stevia packet carry purse little bit xylitol add stevia take away bit stevia yucky bitter taste bit s use sothat

2. great product bad packaging iwould like share experience couscous packaging osem come couscous great cook fast make delicious north african dishesthe packaging come tear tape couscous place amazon box not damage way blame osem amazon send open tape product

3. tough miniature schnauzer love toy inroad bone end hope long

4. like flavor coffee enjoyable fall season know open special occasion opinion pleasantly surprised delightful ask refill tell amazon quizical

5. pretty good instant good misocup pretty alright honestly ve brand freezedrie instant miso soup well defense ve lot misocup brand buy case product benefit expensive despite fact organicthe basic flavor expect red miso hint dryish papery taste mouth feel go away let hydrate long time suspect come packaging especially prevalent piece tofu come despite high sodium content

Samples of 4 star rating

Showing 5 sample reviews for Rating 4:

1. apple cystaltini big fan crystal lite new appletini flavor bit sweet tart fond turn recognizable appletini green water add find add splash ginger ale give zing crave bit effervescencein case like crystal lite add dash traditional vodka quick easy drink dropin guest like zero bartende skillsi like fact come individual tube use entire packet quart pitcher opinion weak use small glass refreshing sour apple kind way buy

2. sure work delicious not test output increase eat cookie delicious stay moist ingredient brewer yeast suppose aid milk production base internet research doctor believe ingredient lead modest gain

3. richtasting fullbodie preferred method presspot aka french press taste nuance coffee bigbodied brew disappoint recommend consume asap camano island coffee roaster correct previous packaging problem

4. john lightne fast delivery nice coffee reasonably price thank order suppose

5. say dressing thick hot spicy promise find dilute little taste make calorie few like flavor turn vegetableinto party eat tomato green bean vegetable cook raw m definitely go buy

Samples of 5 star rating

Showing 5 sample reviews for Rating 5:

1. great hair buy product help condition hair work great cook try smell awesome

2. good butter ve taste amish roll butter good ve great toast bagel roll well baked potatofortunately buy grocery store county market punxsutawney pa s large amish community

3. caribou win hand sample kcup caribou blend good daily regular coffee drinker m huge fan flavor coffee love dark bitter bold type simply want smooth cup joe drink daily feel comfort consistently tasting awesome caribous perfect fit far brewing caribou kcup home keep go starbucks monthit taste well

4. good syrup good prize torani syrup latte home long spend 4 latte drink torani vanilla syrup good market

5. love cereal sure cereal time amazon sale house love little one eat right box healthy

Top 20 unique words and count in each rating

Rating 1:

('worst', 2035), ('horrible', 2044), ('waste', 2164), ('awful', 1896), ('terrible', 1952), ('disgusting', 1016), ('refund', 883), ('garbage', 582), ('gross', 862), ('nasty', 975), ('trash', 563), ('beware', 1034), ('crap', 420), ('ripoff', 278), ('rip', 397), ('dangerous', 298), ('false', 321), ('expired', 441), ('toxic', 270), ('sick', 861)

Rating 2:

('overpaid', 11), ('fleer', 6), ('rumor', 7), ('overtook', 3), ('idol', 3), ('miserly', 3), ('hypocritical', 3), ('dissatisfying', 3), ('sucky', 3), ('dissenter', 3), ('abominable', 3), ('futility', 3), ('emphatic', 3), ('fruitless', 3), ('gibberish', 3), ('sloppily', 4), ('resigned', 4), ('damper', 4), ('breach', 2), ('fluent', 2)

Rating 3:

('undecided', 16), ('troubled', 7), ('tempt', 12), ('stingy', 16), ('unproven', 3), ('glare', 3), ('valiant', 3), ('merrily', 4), ('corrosion', 3), ('torment', 3), ('unsustainable', 3), ('uncomplicated', 4), ('zombie', 12), ('petty', 7), ('reachable', 2), ('insultingly', 2), ('vestiges', 2), ('alienate', 2), ('vibrated', 2), ('aborted', 2)

Rating 4:

('quibble', 29), ('drawback', 215), ('qualm', 15), ('frolic', 5), ('elegantly', 5), ('beggar', 5), ('crumple', 6), ('catchy', 17), ('bleeds', 6), ('imaginative', 6), ('restlessness', 4), ('uplift', 7), ('monotony', 8), ('jarring', 8), ('shimmy', 5), ('obscures', 5), ('quibbles', 5), ('sincerity', 3), ('impair', 3), ('amply', 3)

Rating 5:

('awesome', 1898), ('amazing', 2092), ('heaven', 380), ('addicted', 494), ('fabulous', 477), ('addicting', 166), ('wonderful', 3770), ('awsome', 57), ('versatile', 264), ('excellant', 45), ('smoothest', 47), ('excelent', 32), ('unbeatable', 56), ('sensational', 29), ('superb', 243), ('exquisite', 64), ('marvelous', 65), ('splendid', 35), ('lifesaver', 70), ('exellent', 15)

Logic used to find unique words:

Dominant/Characteristic Words (Relative Frequency)

Definition: Words that are much more frequent in the chosen rating compared to others.

Logic: Compute relative frequency per rating.

$$\text{score}(\text{word}) = f_{\text{target}}(word) - \sum_{\text{other ratings}} f_{\text{other}}(word)$$

$$f_{\text{rating}}(word) = \frac{\text{count}(\text{word in that rating})}{\text{total sentiment words in that rating}}$$

Samples of food product in dataset

'coffee', 'tea', 'chocolate', 'water', 'sugar', 'milk', 'sauce', 'oil', 'snack', 'cookies', 'salt', 'cereal', 'candy', 'rice', 'chicken', 'butter', 'coconut', 'juice', 'beans', 'cheese', 'corn', 'peanut', 'chips', 'soup', 'honey', 'syrup', 'bread', 'cookie', 'pasta', 'ginger', 'cream', 'popcorn', 'crackers', 'cinnamon', 'noodles', 'wheat', 'date', 'soy', 'flour', 'lemon', 'soda', 'espresso', 'beef', 'orange', 'apple', 'cake', 'oatmeal', 'granola', 'pepper', 'snacks', 'fish', 'cherry', 'olive', 'almond', 'potato', 'garlic', 'mint', 'pumpkin', 'peanuts', 'tuna', 'chili', 'strawberry', 'almonds', 'salmon', 'oats', 'candies', 'peach', 'yogurt', 'wine', 'cracker', 'jelly', 'vinegar', 'banana', 'tomato', 'curry', 'hazelnut', 'blueberry', 'eggs', 'raspberry', 'bacon', 'mango', 'lime', 'grape', 'egg', 'pancakes', 'broth', 'pie', 'tart', 'potatoes', 'pizza', 'soups', 'tomatoes', 'mustard', 'pork', 'biscuits', 'turkey', 'beer', 'mocha', 'latte', 'pomegranate', 'onion', 'taco', 'cherries', 'dates', 'cakes', 'sandwich', 'peppers', 'peas', 'cappuccino', 'herbs', 'broccoli', 'salsa', 'cashews', 'blueberries', 'sesame', 'pineapple', 'apples', 'brownies', 'coke', 'strawberries', 'spaghetti', 'wrap', 'cola', 'quinoa', 'muffins', 'onions', 'waffles', 'steak', 'carrots', 'fig', 'watermelon', 'lemonade', 'pancake', 'sandwiches', 'jam', 'donut', 'barley', 'brownie', 'lamb', 'mushrooms', 'sausage', 'pear', 'shrimp', 'pudding', 'spinach', 'duck', 'smoothie', 'toffee', 'mayo', 'ketchup', 'walnuts', 'waffle', 'biscuit', 'rum', 'kiwi', 'blackberry', 'basil', 'chia', 'tortilla', 'bananas', 'marshmallow', 'ham', 'muffin', 'tacos', 'mushroom', 'pea', 'crisps', 'clam', 'peaches', 'crab', 'donuts', 'celery', 'carrot', 'cashew', 'vodka', 'pears', 'walnut', 'wafers', 'cloves', 'lentils', 'pistachios', 'plum', 'pepsi', 'lobster', 'figs', 'nutmeg', 'wafer', 'mayonnaise', 'rosemary', 'raspberries', 'flaxseed', 'parsley', 'burger', 'pickles', 'burgers', 'lemons', 'cardamom', 'pickle', 'clove', 'cilantro', 'pastry', 'lettuce', 'dill', 'hazelnuts', 'oranges', 'paneer', 'kale', 'ghee', 'grapes', 'pistachio', 'whiskey', 'turmeric', 'mangoes', 'cucumber', 'wraps', 'sardine', 'oregano', 'avocado', 'pastries', 'papaya', 'guava', 'gin', 'millet', 'asparagus', 'cauliflower', 'brandy', 'kimchi', 'melon', 'coriander', 'cabbage', 'thyme', 'anchovy', 'nachos', 'oyster', 'chilies', 'blackberries', 'limes', 'zucchini', 'plums', 'squid', 'chutney', 'okra', 'octopus', 'cornflakes', 'milkshake', 'icecream', 'eggplant', 'hotdogs', 'curd', 'biryani', 'radish', 'hotdog', 'sauerkraut', 'prawns', 'turnip', 'beetroot', 'prawn', 'samosa', 'mutton', 'melons', 'dosa', 'brinjal', 'soysauce'

7. Train-Test Split

Train-Test Split is a technique to divide the dataset into two separate sets:

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

This separation ensures that the model generalizes well and is not just memorizing the training data.

Why Stratified Split?

In classification problems like review rating prediction, stratified splitting is important to maintain class balance (equal distribution of ratings) in both training and testing sets.

Without stratification, some rating classes (like 1-star or 5-star) may be underrepresented in the test set, leading to biased evaluation.

How It Was Done:

The dataset was split using the train_test_split() function from Scikit-learn. An 80–20 ratio was applied, where 80% of the data was assigned to the training set and 20% to the testing set. The stratify=df['label'] argument ensured that the distribution of class labels remained consistent across both sets, meaning that each rating category was proportionally represented in the train and test splits. Additionally, a fixed random_state=42 was used to guarantee reproducibility, so that the data split remains the same every time the code is executed.

Code:

```
train_df, test_df = train_test_split(df, test_size=0.2,
stratify=df['label'], random_state=42)
```

8. Conversion to Hugging Face Datasets

The train_df and test_df were converted to datasets.Dataset objects from Hugging Face to support seamless integration with tokenization and training workflows:

Code:

```
from datasets import Dataset

train_dataset = Dataset.from_pandas(train_df) test_dataset

= Dataset.from_pandas(test_df)
```

9. Tokenization for BERT

Used the BERT tokenizer from Hugging Face's Transformers library (bert-base-uncased):

Tokenization included:

- Padding to a fixed length (64 tokens)
- Truncating long sequences

Code:

```
from transformers import AutoTokenizer
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
def tokenize(example):
    return tokenizer(example["text"], truncation=True, padding="max_length", max_length=64)
train_dataset = train_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)
```

11. Formatting for PyTorch

The tokenized datasets were formatted for PyTorch-based training

Code:

```
train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "label"])
```

12. Training the Dataset

Link: https://github.com/rajasak/AUTOMATED-REVIEW-SYSTEM-/blob/main/AI/Training%20the%20dataset.pdf

13.BERT

Link: https://github.com/rajasak/AUTOMATED-REVIEW-SYSTEM-/blob/main/AI/bert.pdf