

# 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
- 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

rajasak Add files via upload		
app	Add empty folders with .gitkeep files	last week
data	Add files via upload	yesterday
frontend	Add empty folders with .gitkeep files	last week
models	Add empty folders with .gitkeep files	last week
notebooks	Add files via upload	yesterday
README.md	Initial commit	last week
requirement.txt	Update requirement.txt	last week

## 4. Data Collection

- Data was collected through web scrapping and from Kaggle
- Data collected from Kaggle had 49000 rows and it was dataset related to cloths
- Data collected from web scrapping had 1700 rows and it was dataset related to iphone 15
- Both the dataset were merged
- Dataset link= <https://github.com/rajasak/automated-review-rating-system/blob/main/data/merged.csv>
- Final dataset contains 2 column with Review and Rating
- 1 star=4196, 2 star= 3447, 3 star= 5695, 4 star=8316 , 5 star= 29294

### 4.1. Balanced dataset

- Link for balanced dataset= [https://github.com/rajasak/automated-review-rating-system/blob/main/data/balanced\\_data.csv](https://github.com/rajasak/automated-review-rating-system/blob/main/data/balanced_data.csv)
- Balanced dataset was created with 2000 rows of each rating

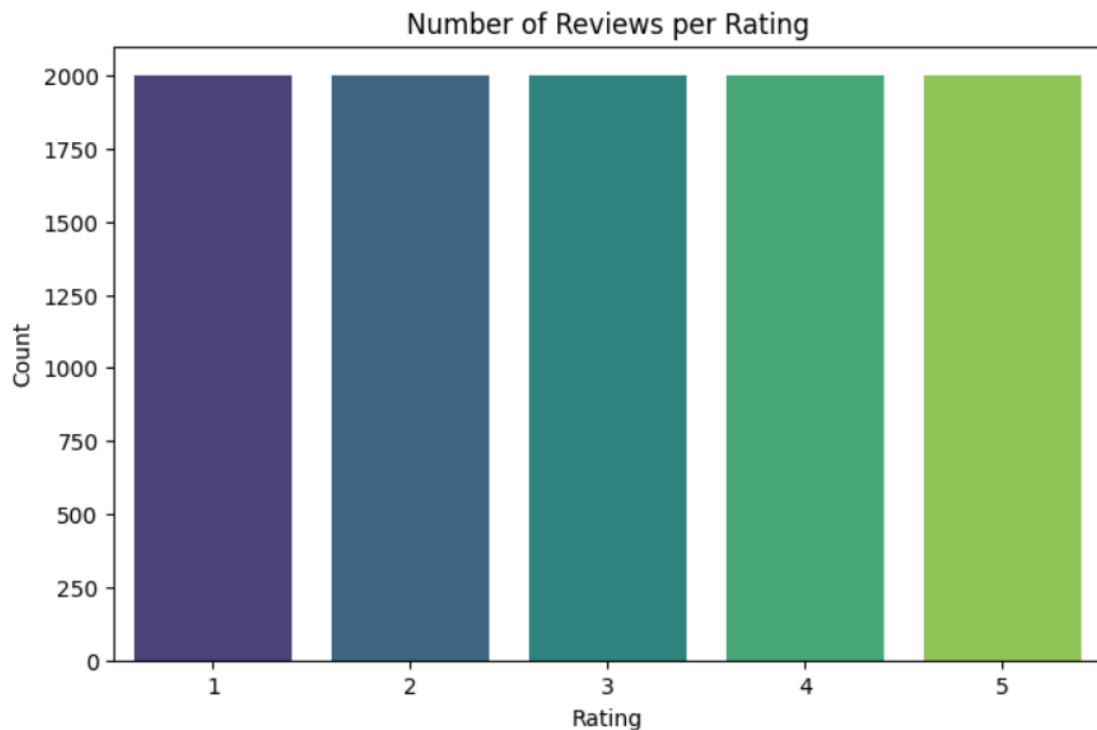


Fig count plot

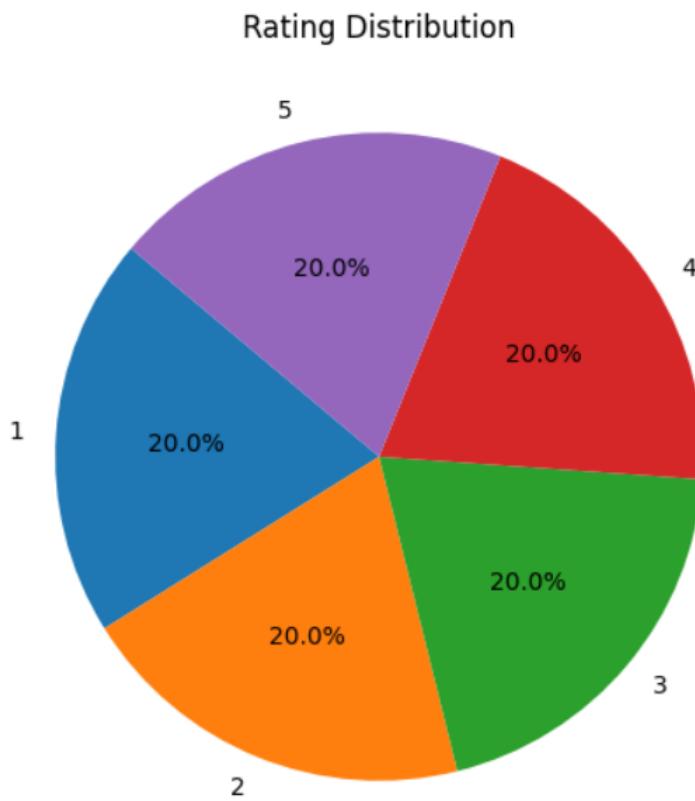


Fig pie chart

## 4.2. imbalanced dataset

- Link for imbalanced dataset= [https://github.com/rajasak/automated-review-rating-system/blob/main/data/imbalanced\\_ratio\\_dataset3.csv](https://github.com/rajasak/automated-review-rating-system/blob/main/data/imbalanced_ratio_dataset3.csv)
- Imbalanced dataset was created with 1 star=10%, 2 star=15%, 3 star=15%, 4 star=30%, 5 star=20%

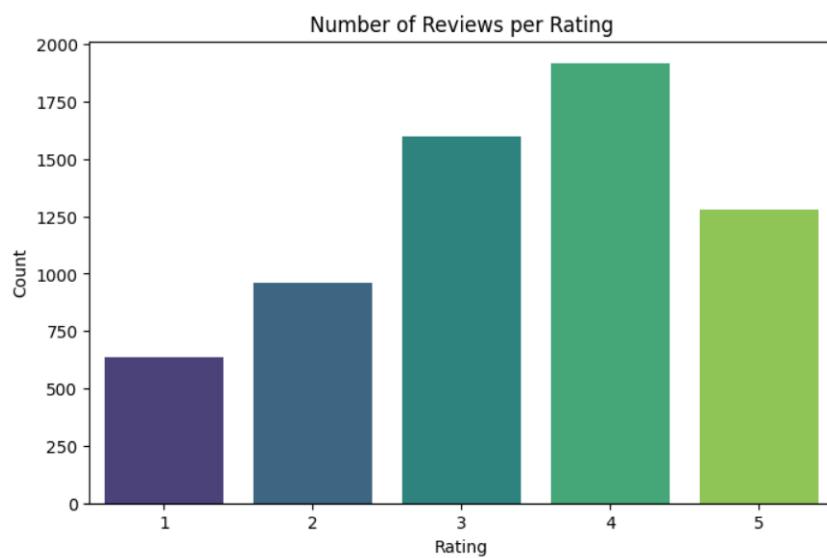


Fig count plot

---

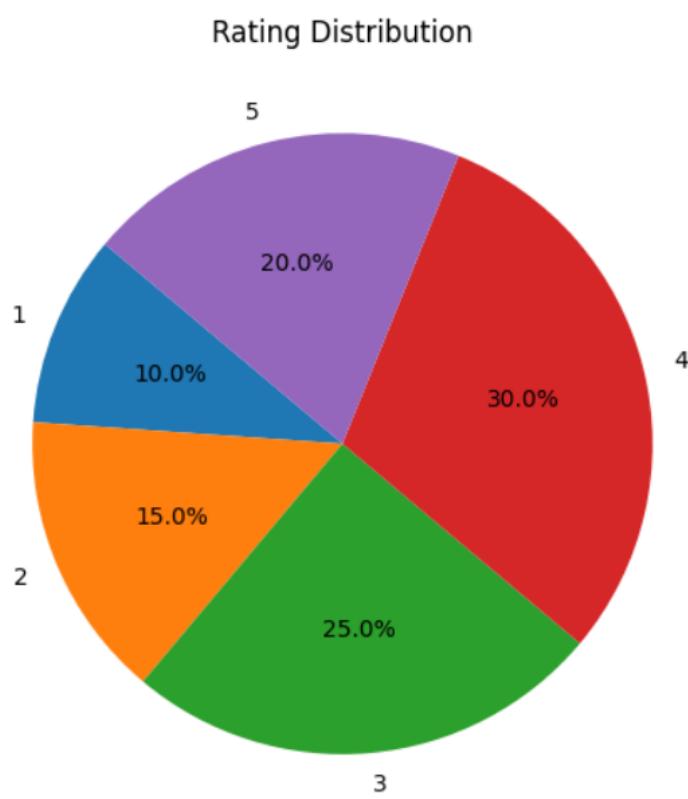


Fig pie chart

# 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:

```
In [4]: has_duplicates = df.duplicated().any()  
print(has_duplicates)
```

```
True
```

```
In [5]: num_duplicates = df.duplicated().sum()  
print(f"Number of duplicate rows: {num_duplicates}")
```

```
Number of duplicate rows: 12404
```

```
In [6]: df.drop_duplicates(inplace=True)
```

```
In [7]: num_duplicates = df.duplicated().sum()  
print(f"Number of duplicate rows: {num_duplicates}")
```

```
Number of duplicate rows: 0
```

## 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:

```
In [9]: # 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])

Number of conflicting review texts: 1484
```

```
In [10]: conflicting_reviews = df.groupby('Review')['Rating'].nunique()
conflicting_reviews = conflicting_reviews[conflicting_reviews > 1].index
df = df[df['Review'].isin(conflicting_reviews) == False]
```

```
In [11]: conflicting_counts = df.groupby('Review')['Rating'].nunique()
conflicting_reviews = conflicting_counts[conflicting_counts > 1]
print("Number of conflicting review texts:", conflicting_reviews.shape[0])

Number of conflicting review texts: 0
```

## 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:

```
In [4]: df = df.dropna(subset=['Rating'])
```

## 5.4 Dropping Unnecessary Columns

Non-essential columns such as review IDs, timestamps, 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:

```
In [47]: #removing punctuations and making Lower case
X_train = X_train.str.lower()
X_train = X_train.str.replace(r'\[.*?\]', '', regex=True)

In [48]: import re

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 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', 'am', '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', 'besides', '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', 'everywhere', 'except', 'few', 'fifteen', 'fifty', 'first', 'five', 'for', 'former', 'formerly', 'forty', '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', 'its', 'itself', 'just', 'keep', 'last', 'latter', 'latterly', 'least', 'less', 'made', 'make', 'many', 'may', 'me', 'meanwhile', 'might', 'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'much', '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', 'quiet', 'rather', 're', 'really', 'regarding', 'same', 'say', 'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'take', 'ten', 'than', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'third', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two', 'under', 'unless', 'until', 'up', 'upon', 'us', 'used', 'using', 'various', 'very', 'via', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever', 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves', 'd', 'll', 'm', 're', 's', 've', 'd', 'll', 'm', 're', 's', 've']
```

Code:

```
In [52]: #Lemmatization and stop word removal
def cleaning(text):
    doc = nlp(text)
    return ' '.join([token.lemma_ for token in doc if not token.is_stop])
```

```
In [54]: X_train=X_train.apply(cleaning)
```

## 5.10 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.

## 5.11 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 100 words.

This ensures that the dataset remains robust and relevant for model training.

# 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.

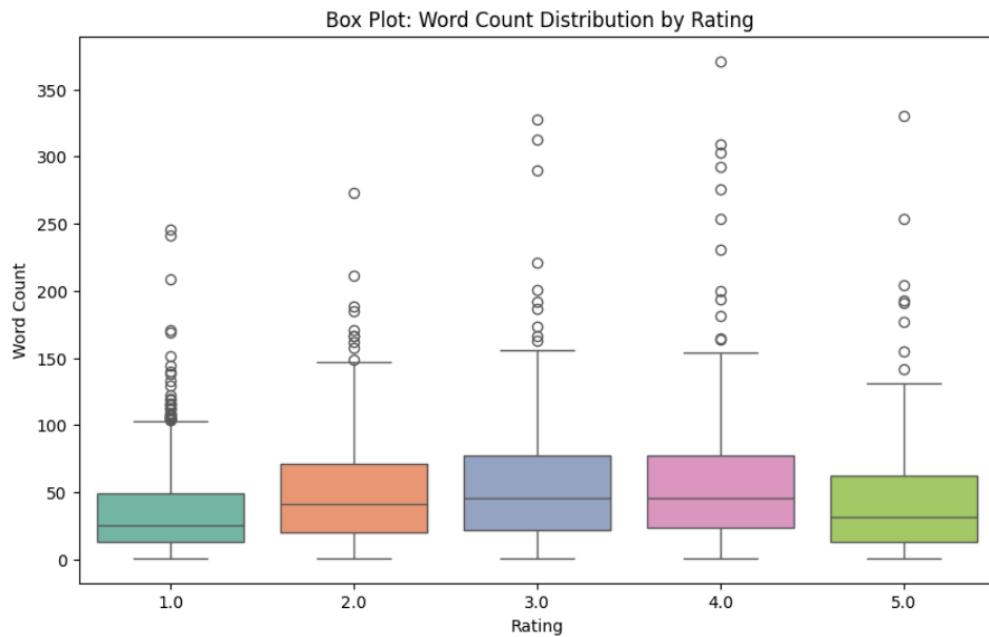


Fig box plot

## 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.

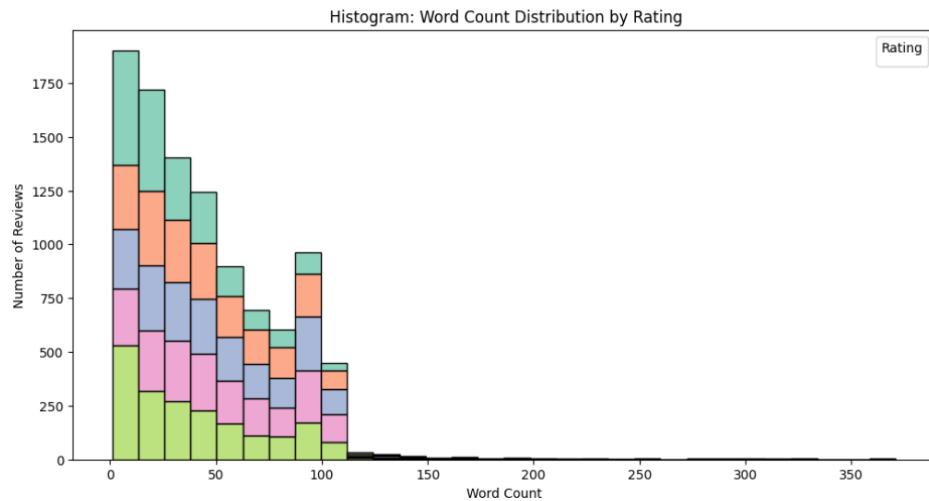


Fig histogram

## Barplot

A **bar plot** is a chart that represents categorical data with rectangular bars, where the height of each bar indicates the value or frequency of that category. In this project, bar plots were used to show the number of reviews per rating class (1 to 5 stars), helping to visualize class distribution and detect any imbalance in the dataset.

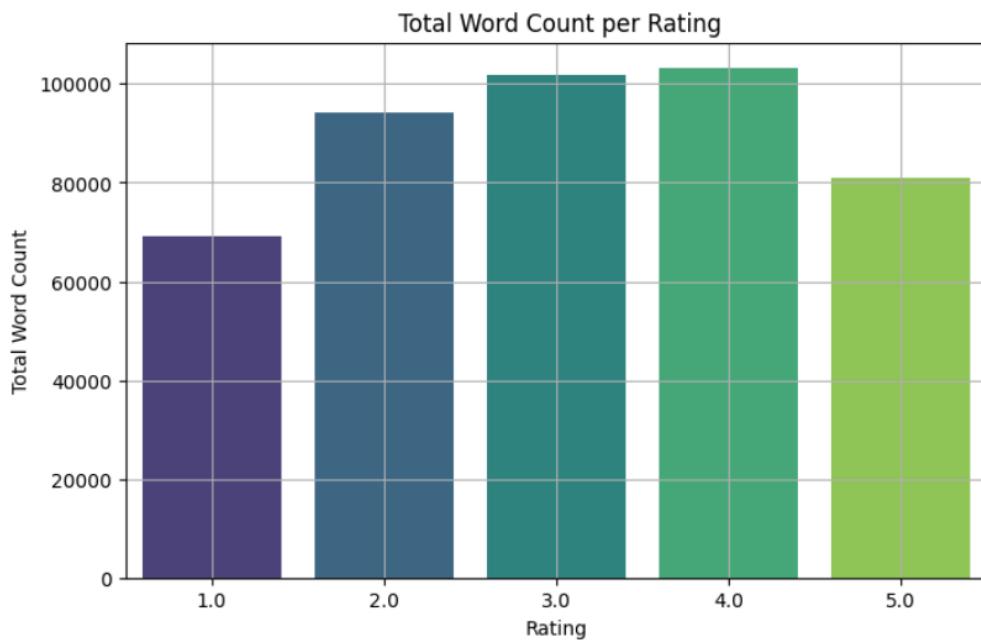


Fig bar plot

## **Samples of 1 star rating:**

Showing 5 sample reviews for Rating 1:

1. This shirt didn't fit correctly so I returned it. I dropped it off at UPS 8/2 and they never refunded me.
2. I really liked this shirt the one time i got to wear it...but when i washed it, according to the directions on the tag, it shrunk! a lot ! the sleeves are now too tight, and it is much shorter. thinking about returning it to the store, but in any case, i cannot wear it again.
3. I have the purple jacket is short on sleeves and pants were short to very disappointed I have the black,red and grey and they fitted perfectly I don't understand why this purple suit was made like this and they are all the same size smh
4. Oh I am so, so, sad. I thought I had found the perfect jeans. I bought a pair about a year ago and loved them. So, I ordered these. Imagine my surprise when I realized there were no front pockets. I mean. Really? I do not want to carry a purse or wear more layers of clothes. I want to feel skinny in a pair of jeans, and have pockets. Disappointing.
5. It was too big for my daughter, I returned it from UPS and so far I haven't received the credit to get another one.

## **Samples of 2 star rating**

Showing 5 sample reviews for Rating 2:

1. 🙄
2. Material to heavy for summer.
3. Based on the other reviews, i sized down from my usual 6p to a 2p, but this dress was still ridiculously huge and extremely unflattering. i could tell the minute i took it out of the package it wouldn't work. i can't imagine this would flatter very many people. the front photo of the model is incredibly deceptive. very disappointed--this will be going back for sure.
4. Advertised as loose fit, but definitely snug and way too short.
5. Not too fond of the material and the way it looks like a rain coat

## **Samples of 3 star rating**

Showing 5 sample reviews for Rating 3:

1. Pants were fine but pockets ripped immediately. Only buy if you know how to sew.
2. I love the fabric however the overall fit did not work for me.  
i think the arms are much smaller than the overall shirt.
3. The look and feel of the swim trunks are great but the quality should be higher for the price. After one wash and wear the waist band stitching started to come loose. I would expect some wear and tear after several uses but not after one.
4. Did not like fabric, does not drape well.
5. The wrong color was sent to me. I kept them anyways, but I'm not crazy about the fit of these. The tag on the outside waistband somehow digs up into my skin when i sit, making it mildly uncomfortable. They are a nice Jean quality though.

## **Samples of 4 star rating**

Showing 5 sample reviews for Rating 4:

1. The fabric is nice and the color is a rich jewel-tone. i have a couple of very similar shirts from a similar store that are several years old. those fit a bit more loss sly around the stomach (not baggy, just body-skimming) which made them more flattering for more body types. this shirt works if you have a little padding because of the gathered folds, but it probably wouldn't be great for someone carrying a lot of extra weight around their middle. the knot also has a tendency to twist exposing t
2. Style is perfect, year around wear.
3. Bought these in 28 had to return for a 27p . love them!! i order the 2 other colors
4. I really love the look of this sweater and it seems to be of great quality. i will say the fit is a bit odd. the sweater is on the short side but is very roomy through the neck and shoulders. i ordered a small (5'6"; 120lbs) and would change for an xs but i'm afraid it would be cropped. i'm keeping it but wish it had a more uniform fit.  
really beautify workmanship though.
5. These jeans for great and were flattering. Perfect length for me which is always a problem I have with high waisted pants. They were long enough but didn't drag under foot. I love the double buttons!! Good quality jeans all around... good purchase!!

## **Samples of 5 star rating**

Showing 5 sample reviews for Rating 5:

1. Very comfortable and fits well.
2. This top is really pretty and nice quality. runs big - i went down a size, and its perfect. coloring is more subtle in person than in the photo.
3. Happy with fit and quality
4. As a full time photographer, I am always looking for breathable, cute and professional shirts to wear to sessions and weddings. I love this shirt, and it's honestly perfect for every occasion. Time to go buy it in every color lol
5. I love how versatile this dress is! Listen to the reviews as if you do have a little bit of a belly or pooch, def size up. I'm normally a 4, 5'9 and weight 150 pounds. I ordered the 8-10 and I'm so glad I did. Fits very well- may have to wear spanx only bc I do not have a flat stomach. If you do, order your normal size. Fits very well in the chest and the rest of my body.

## 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:

To prepare the data for model training, the dataset was first **shuffled randomly** to eliminate any order bias. A **stratified train-test split** was then performed using `train_test_split()` from `sklearn.model_selection` with the `stratify=y` argument to ensure that all star ratings were **proportionally represented** in both training and testing sets. The dataset was split using an **80% training and 20% testing ratio**. After splitting, all **text preprocessing steps**—including lowercasing, lemmatization, stopword removal, and cleaning—were applied **separately** to `x_train` and `x_test` to **prevent data leakage** and maintain model integrity.

## Code:

### train and test split

```
In [22]: # Shuffle the balanced DataFrame  
df = df.sample(frac=1, random_state=42).reset_index(drop=True)  
  
In [23]: from sklearn.model_selection import train_test_split  
  
In [24]: x=df['Review']  
y=df['Rating']  
  
In [25]: X_train, X_test, y_train, y_test = train_test_split(  
        x, y, test_size=0.2, random_state=42, stratify=y  
)
```

## 8. Text Vectorization

Machine learning models can't work directly with raw text—they require numerical input. **Vectorization** is the process of converting text data into numerical features.

### TF-IDF (Term Frequency - Inverse Document Frequency)

**TF-IDF** is a statistical technique that represents how important a word is to a document relative to the entire corpus.

- **TF** (Term Frequency): How often a word appears in a document.
- **IDF** (Inverse Document Frequency): Penalizes common words and highlights rare but important ones.

This approach helps to capture both **word relevance** and **discriminative power**, making it better than simple word counts.

## Formula for TF-IDF:

$$TF(t, d) = \frac{\text{number of times } t \text{ appears in } d}{\text{total number of terms in } d}$$

$$IDF(t) = \log \frac{N}{1 + df}$$

$$TF - IDF(t, d) = TF(t, d) * IDF(t)$$

## Code:

### TF-IDF

```
In [66]: from sklearn.feature_extraction.text import TfidfVectorizer

In [69]: # create a tfidf vectorizer matrix
tv = TfidfVectorizer()
Xtrain = tv.fit_transform(X_train)

In [70]: X = pd.DataFrame(Xtrain.toarray(), columns=tv.get_feature_names_out())
X
```

	00	000	00p	04	06	0p	Or	10	100	100lbs	...	zipper	zippered	zippers	zipping	zips	zity	zone	zoolander	zoor
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.217494
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.136514	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7491	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
7492	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
7493	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
7494	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
7495	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

7496 rows × 8816 columns

## 9. Training the model

### Algorithm: Logistic Regression

Logistic Regression is a supervised learning algorithm used primarily for classification tasks. In this project, it is used for multi-class text classification to predict review ratings (1–5 stars) based on review content. Despite its name, Logistic Regression is a classification, not regression, algorithm.

### How the Algorithm Works

#### Binary Classification

For binary classification, logistic regression models the probability that an input  $\mathbf{x}$  belongs to class 1:

$$z = \mathbf{w}^T \mathbf{x} + b$$

$$P(y = 1 | \mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Where:

- $w$ : weight vector
- $x$ : feature vector (e.g., TF-IDF)
- $b$ : bias term
- $\sigma(z)$ : sigmoid function that maps  $z$  to  $(0, 1)$

Prediction is made by:

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

## Multi-Class Classification (Softmax Regression)

When there are more than two classes (e.g., 5 review rating levels), logistic regression uses the **softmax function** to compute the probability of each class  $k \in \{1, 2, \dots, K\}$ :

$$z_k = \mathbf{w}_k^T \mathbf{x} + b_k$$
$$P(y = k | \mathbf{x}) = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } k = 1, 2, \dots, K$$

The model predicts the class with the **highest probability**:

$$\hat{y} = \arg \max_k P(y = k | \mathbf{x})$$

### Key Hyperparameter:

Hyperparameter	Description	Default	Tuning Tips
C	Inverse of regularization strength	1.0	Lower = stronger regularization
penalty	Type of regularization ( $l_1$ , $l_2$ , etc.)	$l_2$	Use $l_2$ for text data
solver	Optimization algorithm (liblinear, saga, etc.)	lbfgs	Use saga for $l_1$ or large datasets
multi_class	Strategy: ovr (one-vs-rest) or multinomial (softmax)	auto	Use multinomial with softmax logic
max_iter	Max optimization iterations	100	Increase if not converging

### Strengths:

- Fast training and prediction
- Works well with high-dimensional sparse data (like TF-IDF)
- Easy to interpret (coefficients show feature importance)
- Performs well on linearly separable classes

### **Limitations:**

- Assumes linear decision boundary
- Not ideal for complex, nonlinear relationships
- Sensitive to irrelevant features and multicollinearity
- Performance drops with highly imbalanced classes

### **Use When:**

- Dataset is large and well-preprocessed
- You need a fast, interpretable baseline
- Input features are TF-IDF or BoW vectors

### **Avoid When:**

- Data is highly nonlinear or hierarchical
- High accuracy is required for long-term deployment
- You want contextual embeddings (BERT, LSTM, etc.)

### **Model Evaluation: Logistic Regression Results:**

Code:

```
# Train Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(xtrain_vec, y_train)

# Predict
y_pred_lr = lr.predict(xtest_vec)

# Evaluation
print("❖ Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

Result:

Logistic Regression:				
Accuracy: 0.44718864950078824				
Classification Report:				
	precision	recall	f1-score	support
1.0	0.49	0.56	0.52	371
2.0	0.38	0.33	0.35	382
3.0	0.36	0.32	0.34	383
4.0	0.40	0.43	0.42	387
5.0	0.58	0.60	0.59	380
accuracy			0.45	1903
macro avg	0.44	0.45	0.44	1903
weighted avg	0.44	0.45	0.44	1903

After training the Logistic Regression model on the text data using TF-IDF features, we evaluated its performance on the test set using standard classification metrics.

- **Accuracy:** 0.447 (i.e., ~44.7%)  
This indicates that the model correctly predicted the review class in about 45% of the test samples, which is a reasonable baseline for a 5-class classification task.
- **Macro Average F1-Score:** 0.44  
A simple average over all classes, treating each class equally.
- **Weighted Average F1-Score:** 0.44  
Accounts for the support (number of instances) of each class, giving a more balanced overall view.
- The model performs **best on class 5.0**, with an F1-score of **0.59**, indicating it is best at identifying highly positive reviews.
- Classes **2.0 and 3.0** have the **lowest performance**, which suggests overlap in feature space or ambiguity in the reviews that correspond to neutral sentiments.