

Automated Review Rating System

1. Project Overview

The project **Automated Review Rating System** aims to develop a smart framework capable of predicting star ratings from customer reviews. Using natural language processing techniques, the system interprets textual feedback to identify sentiment and linguistic patterns. Machine learning models are trained on labeled review data to establish relationships between customer opinions and corresponding rating levels. By evaluating performance across different dataset distributions, the system demonstrates the effectiveness of AI-driven solutions in providing consistent, scalable, and efficient review analysis.

2. Environment Setup

The project was developed and executed in **Jupyter Notebook**, providing an interactive environment for code execution and analysis. The implementation was carried out using **Python 3.10**, along with several essential libraries for data preprocessing, visualization, and machine learning.

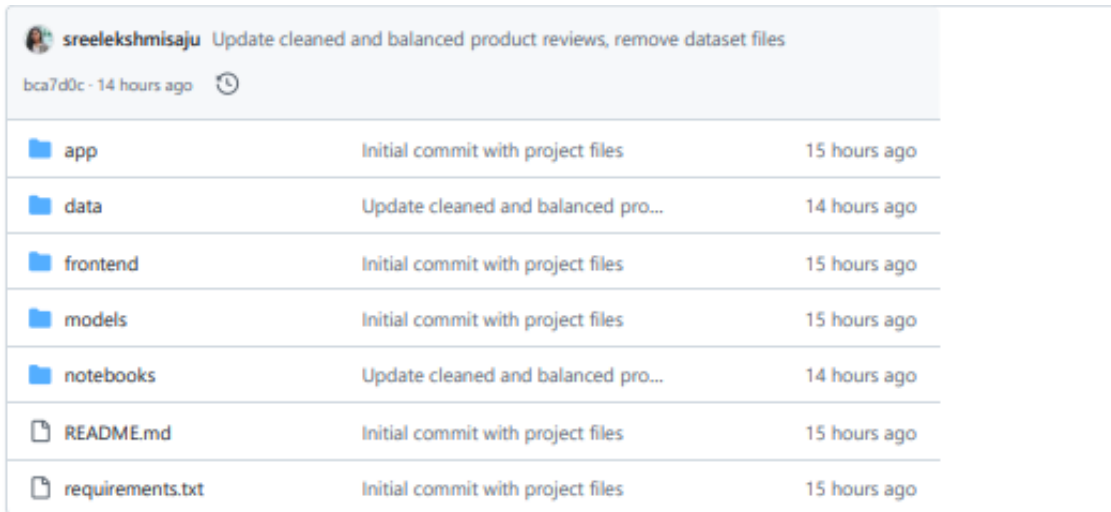
The following libraries were installed and utilized:

- **pandas** – for dataset loading, manipulation, and analysis.
- **numpy** – for numerical computations and efficient array handling.
- **matplotlib** & **seaborn** – for data visualization and graphical representation of trends.
- **scikit-learn** – for feature extraction, model training, evaluation, and performance metrics.
- **spaCy** – for natural language processing and lemmatization.
- **re** – for text pattern recognition and cleaning using regular expressions.

3. GitHub Project Setup

A GitHub repository named Automated-Review-Rating-System was created to manage the project's code, notebooks, and resources. The repository provides version control, collaboration capabilities, and a centralized location for all project files.

3.1 Directory Structure



sreelekshmisaju Update cleaned and balanced product reviews, remove dataset files bca7d0c - 14 hours ago		
app	Initial commit with project files	15 hours ago
data	Update cleaned and balanced pro...	14 hours ago
frontend	Initial commit with project files	15 hours ago
models	Initial commit with project files	15 hours ago
notebooks	Update cleaned and balanced pro...	14 hours ago
README.md	Initial commit with project files	15 hours ago
requirements.txt	Initial commit with project files	15 hours ago

Figure_1: GitHub Repository Setup

4. Data Collection

The project utilizes publicly available datasets from Kaggle to train and evaluate the Automated Review Rating System. Two datasets were collected:

- **harvard_reviews.csv** – contains 3,184 reviews across 7 columns. [Dataset Link](#)
- **yelp.csv** – contains 10,000 reviews across 10 columns. [Dataset Link](#)

After merging both datasets, the final merged dataset contains 13,013 reviews. The distribution of ratings in the merged dataset is as follows:

Rating	Count
1	761
2	970
3	1,711
4	4,477
5	5,094

Table 1: Distribution of ratings in the merged dataset

The cleaned and merged dataset is available at the following link: [Merged Dataset Link](#)

This combined dataset provides a robust foundation for training machine learning models to predict review ratings.

4.1 Balanced Dataset

Balanced dataset ensures that each rating category is equally represented, preventing bias during model training. The balanced dataset used contains 10,000 reviews with 2,000 reviews for each rating level. Maintaining equal representation across ratings helps the model learn patterns from all categories fairly and improves prediction reliability.

The dataset is available at the following link: [Balanced Dataset Link](#)

Balanced Dataset Details:

```
Balanced Dataset Distribution:
```

```
Rating
```

```
2    2000
```

```
3    2000
```

```
5    2000
```

```
4    2000
```

```
1    2000
```

```
Name: count, dtype: int64
```

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

5. Data Preprocessing

Data preprocessing is a critical step in preparing raw text data for machine learning. It involves cleaning, standardizing, and transforming the data to ensure it is consistent, meaningful, and suitable for analysis. These steps help improve model accuracy and reliability by removing noise, normalizing text, and structuring the reviews for effective feature extraction.

5.1 Concatenating Datasets

Since reviews were collected from two different sources (`harvard_reviews.csv` and `yelp.csv`), both datasets were combined to form a single dataset for model training and analysis. This ensures a larger and more diverse collection of reviews, enhancing the model's ability to generalize across different domains.

Operation Used:

- `pd.concat([df, df1], axis=0)` was applied to merge the datasets vertically (row-wise).
- `.reset_index(drop=True)` was used to reassign continuous indexing after concatenation.

Purpose:

- To create a larger and more diverse dataset.
- To improve the generalizability and robustness of the model during training and evaluation.

Python Implementation:

```
df2 = pd.concat([df, df1], axis=0).reset_index(drop=True)
print("Combined dataset shape:", df2.shape)
df2.head()
```

This combined dataset serves as the final input for preprocessing and subsequent model development.

5.2 Renaming Columns

Column names are standardized to improve clarity and make the dataset easier to work with. For example, columns containing review text and ratings are renamed to Review and Rating, respectively. Standardized column names help avoid confusion during analysis and make the code and documentation more readable and consistent.

The following Python code was used to rename dataset columns for consistency:

```
df.rename(columns={'text': 'Review', 'rating': 'Rating'}, inplace=True)
df1.rename(columns={
    "stars": "Rating",
    "text": "Review"
}, inplace=True)

df1 = df1[[ "Rating", "Review"]]
```

5.3 Count Missing Values

Missing or null values in the dataset are identified and counted for each column. This helps assess data completeness and determine if any records require removal or correction to maintain data quality.

The following Python code was used to count missing values in the dataset:

```
missing_data = df2.isnull().sum()
print("\nMissing Data Count per Column:")
print(missing_data)
print("\nTotal Missing Values in Dataset:", missing_data.sum())
```

Output:

```
Missing Data Count per Column:
ï»¿"published_date"    10000
published_platform    10000
Rating                0
type                  10000
helpful_votes         10000
title                 10000
Review                0
dtype: int64

Total Missing Values in Dataset: 50000
```

5.4 Handle Missing Data

Records with missing or incomplete information are removed or corrected to ensure dataset consistency. Proper handling of missing data maintains the integrity and reliability of subsequent

analysis and modeling.

The following Python code was used to identify and handle missing values in the dataset:

```
print("Missing values in Review and Rating:")
print(df2[['Review', 'Rating']].isnull().sum())
df2.dropna(subset=['Review', 'Rating'], inplace=True)
df2.reset_index(drop=True, inplace=True)
print("\nAfter handling missing data:")
print(df2[['Review', 'Rating']].isnull().sum())
print("Dataset shape:", df.shape)
df2.info()
df2.head()
```

Output:

```
Missing values in Review and Rating:
Review      0
Rating      0
dtype: int64

After handling missing data:
Review      0
Rating      0
dtype: int64
Dataset shape: (13184, 7)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13184 entries, 0 to 13183
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   i»¿"published_date"    3184 non-null  object
1   published_platform     3184 non-null  object
2   Rating                 13184 non-null int64
3   type                   3184 non-null  object
4   helpful_votes          3184 non-null  float64
5   title                  3184 non-null  object
6   Review                 13184 non-null object
dtypes: float64(1), int64(1), object(5)
memory usage: 721.1+ KB
```

5.5 Count Number of duplicates

Duplicate records are identified by checking for repeated entries in the dataset. Counting duplicates helps assess data redundancy and ensures that each observation contributes unique information for analysis and modeling.

The following Python code was used to identify and view duplicate reviews in the dataset:

```
duplicate_count = df2.duplicated(subset=['Review', 'Rating']).sum()
print(f"Number of duplicates: {duplicate_count}")
duplicates = df2[df2.duplicated(subset=['Review', 'Rating'], keep=False)]
print("Sample duplicate rows:")
print(duplicates.head())
```

Output:

```

Number of duplicates: 2
Sample duplicate rows:
   id "published_date" published_platform Rating type helpful_votes \
3468      NaN      NaN      NaN      5 NaN      NaN
7105      NaN      NaN      NaN      5 NaN      NaN
7556      NaN      NaN      NaN      2 NaN      NaN
12864     NaN      NaN      NaN      2 NaN      NaN

   title
3468  NaN      Great service
7105  NaN      Great service
7556  NaN  This review is for the chain in general. The l...
12864  NaN  This review is for the chain in general. The l...

```

5.6 Removing Duplicates and Conflicting Reviews

Duplicate records are removed from the dataset to prevent redundancy and avoid bias in analysis or model training. Ensuring each entry is unique improves the quality and reliability of the data. To ensure data quality and consistency, the following criteria were applied:

- **Exact Duplicates:** Remove records where both the review text and rating are identical. This prevents redundant data that does not add any new information.
- **Conflicting Reviews:** Identify reviews with the same text but different ratings. Remove these conflicting entries to avoid inconsistency in model training.
- **Index Reset:** After removal of duplicates and conflicts, the dataset index is reset to maintain proper sequence and avoid gaps in indexing.

These steps ensure that each review in the dataset is unique, consistent, and reliable for analysis.

The following Python code was used to clean the dataset by removing duplicates and conflicting reviews:

```

df2 = df2.drop_duplicates(subset=['Review', 'Rating'], keep='first')
conflict_idx = df2[df2.duplicated(subset=['Review'], keep=False) &
~df2.duplicated(subset=['Review', 'Rating'], keep=False)
].index
df2.drop(index=conflict_idx, inplace=True)
df2.reset_index(drop=True, inplace=True)
print("After removing duplicates and conflicts:", df2.shape)
df2.head()

```

Output:

```

After removing duplicates and conflicts: (13182, 7)
   id "published_date" published_platform Rating type helpful_votes title Review
0   2023-12-28T08:02:14-05:00      Mobile      5 review      0.0      Best classes and good environment      Good thanks for everything good work group
1   2023-12-12T00:38:26-05:00      Desktop      4 review      0.0      Harvard University      Harvard University was founded in 1636 and is ...
2   2023-12-10T08:21:35-05:00      Mobile      3 review      0.0      Walk around campus      We did a walk around most of the Harvard Campu...
3   2023-11-07T21:20:21-05:00      Desktop      4 review      0.0      Interesting Harvard University      We had a walk through the university grounds w...
4   2023-10-02T17:40:42-04:00      Desktop      5 review      0.0      Lovely university campus has many historic and...      Finally made it to Harvard!! This iconic univ...

```

5.7 Selecting Columns

Only relevant columns required for analysis are retained, reducing dataset complexity and focusing on the essential information.

The following Python code was used to Selecting Columns:

```
df2 = df2[['Rating', 'Review']]
print("Final dataset columns:", df2.columns)
print("Final dataset shape:", df2.shape)
```

Output:

```
Final dataset columns: Index(['Rating', 'Review'], dtype='object')
Final dataset shape: (13182, 2)
```

5.8 Grouping and Aggregation

Data is grouped by relevant attributes, and aggregation functions are applied to summarize or consolidate information for better analysis. The dataset is analyzed to understand its distribution and statistical properties:

- **Count of Reviews per Rating:** Reviews are grouped by the `Rating` column, and the number of reviews in each category is counted. This helps identify class distribution and potential imbalances.
- **Average Rating:** The mean rating is calculated to provide an overall sense of customer sentiment.
- **Minimum and Maximum Ratings:** The dataset's rating range is determined to ensure all ratings fall within the expected bounds (1 to 5).

The following Python code was used to perform these aggregation steps:

```
review_count_per_rating = df2.groupby('Rating')['Review'].count()
print("Number of Reviews per Rating:\n", review_count_per_rating)
average_rating = df2['Rating'].mean()
print("\nAverage Rating of Dataset:", average_rating)
min_rating = df2['Rating'].min()
max_rating = df2['Rating'].max()
print("Rating Range: {} to {}".format(min_rating, max_rating))
```

Output:

```

Number of Reviews per Rating:
Rating
1      763
2      972
3     1725
4     4534
5     5188
Name: Review, dtype: int64

Average Rating of Dataset: 3.9415870125929295
Rating Range: 1 to 5

```

5.9 Cleaning text - Remove emojis, Special characters and Symbols

Text is cleaned by removing emojis, special characters, and symbols to ensure consistency and eliminate noise that could affect analysis. The main steps are:

- **Remove Special Characters and Emojis:** All non-alphanumeric characters, including emojis and symbols, are removed from the text.
- **Normalize Spaces:** Extra spaces, tabs, or line breaks are replaced with a single space, and leading/trailing spaces are removed.

These steps ensure that the review text is consistent, clean, and ready for further preprocessing or feature extraction.

The following Python code was used for text cleaning:

```

import re
def clean_text(text):
    text = re.sub(r'[^A-Za-z0-9\s]+', '', str(text))
    text = re.sub(r'\s+', ' ', text).strip()
    return text
df2.loc[:, 'Review'] = df2['Review'].apply(clean_text)
df2['Review'].head()

```

Output:

```

0    Good thanks for everything good work group har...
1    Harvard University was founded in 1636 and is ...
2    We did a walk around most of the Harvard Campu...
3    We had a walk through the university grounds w...
4    Finally made it to Harvard This iconic univers...
Name: Review, dtype: object

```

5.10 Remove Very Short Reviews

Reviews that are too short often lack meaningful information and may not contribute effectively to analysis or model training. To address this, the following steps were applied:

- **Minimum Length Filter:** Reviews with fewer than 10 characters are removed from the dataset.
- **Index Reset:** After removal, the dataset index is reset to maintain sequential ordering.

This ensures that only informative and substantial reviews are retained for further processing.

The following Python code was used:

```
df2 = df2[df2['Review'].str.len() >= 10]
df2.reset_index(drop=True, inplace=True)
print("After removing very short reviews:", df2.shape)
```

Output:

```
After removing very short reviews: (13014, 2)
```

5.11 Detecting Consecutive Duplicate Words

To ensure textual consistency, reviews were analyzed for consecutive duplicate words such as “*good good*” or “*very very*”. These repetitions often arise due to typing errors or redundancy and may affect the quality of text analysis.

Regex Pattern:

- The regular expression `\b(\w+)(\1\b)+` is used.
- `\b(\w+)` matches a single word.
- `(\1\b)+` ensures the same word appears consecutively.

Function:

- A custom function `find_repeated_words(text)` extracts all repeated word sequences from the review text.

Application:

- The function is applied to each review in the dataset.
- A new column `Repeated_Words` is added to store detected duplicate sequences.
- Reviews containing repetitions are filtered for further inspection.

Purpose:

- Identifies redundant patterns that may reduce clarity in text.
- Supports preprocessing by removing unnecessary repetitions for cleaner data.

Python Implementation:

```
import re
def find_repeated_words(text):
    return re.findall(r'\b(\w+)(\1\b)+', text, flags=re.IGNORECASE)
df2['Repeated_Words'] = df2['Review'].apply(find_repeated_words)
df2_with_duplicates = df2[df2['Repeated_Words'].str.len() > 0]
df2_with_duplicates[['Review', 'Repeated_Words']].head()
```

Output:

	Review	Repeated_Words
79	We had a lovely waking tour of the Harvard cam...	[(Luke, Luke)]
170	I was pleasantly surprised by the lovely museu...	[(there, There)]
268	On a hot hot day we took an Uber to explore Ca...	[(hot, hot)]
439	Came to Harvard for a 3day business conference...	[(etc, etc)]
519	Alot to see We visited the JFK exhibit what a ...	[(to, to)]

5.12 Removing Consecutive Duplicate Words

Repeated consecutive words in the text are removed to standardize the text and improve the quality of textual features for analysis. Consecutive duplicate words in reviews can introduce noise and affect text analysis. To address this, the following steps were applied:

- **Identify Repeated Words:** Patterns where the same word appears consecutively are detected using regular expressions.
- **Remove Repeats:** Consecutive duplicates are reduced to a single occurrence of the word, preserving the meaning while improving text quality.

This step ensures that the review text is clean, standardized, and ready for feature extraction.

The following Python code was used:

```
import re
def remove_repeated_words(text):
    return re.sub(r'\b(\w+) (\1\b)+', r'\1', text, flags=re.IGNORECASE)
df2['Review'] = df2['Review'].apply(remove_repeated_words)
df2.head()
```

Output:

	Rating	Review	Repeated_Words
0	5	Good thanks for everything good work group har...	[]
1	4	Harvard University was founded in 1636 and is ...	[]
2	3	We did a walk around most of the Harvard Campu...	[]
3	4	We had a walk through the university grounds w...	[]
4	5	Finally made it to Harvard This iconic univers...	[]

5.13 Sorting Data

Sorting organizes the dataset based on a specific criterion, making it easier to analyze and process:

- **Sort by Rating:** The dataset is sorted in descending order of the `Rating` column. This helps quickly identify higher-rated reviews and understand rating distribution patterns.
- **Reset Index:** After sorting, the dataset index is reset to maintain sequential order and consistency.

Sorting facilitates structured analysis and subsequent preprocessing steps.

The following Python code was used:

```
df2_sorted = df2.sort_values(by='Rating', ascending=False).reset_index(
    drop=True)
df2_sorted.head()
```

Output:

	Rating	Review
0	5	Yes I do rock the hipster joints I dig this pl...
1	5	The worst Thai food Ive ever had This place wa...
2	5	Finally made it to Harvard This iconic univers...
3	5	This place sucks I moved to the valley and hav...
4	5	This place needs to close their doors get thei...

5.14 Lowercasing Text and Selecting Relevant Columns

Standardizing text to lowercase ensures uniformity and prevents duplicate representations of the same word due to capitalization. The following steps were applied:

- **Select Relevant Columns:** Only the `Rating` and `Review` columns are retained for analysis.
- **Convert to Lowercase:** All review text is converted to lowercase to treat words like “Good” and “good” as the same token.
- **Reset Index:** The dataset index is reset to maintain sequential order after preprocessing.

Lowercasing helps reduce variability in the text and improves the effectiveness of subsequent analysis and feature extraction.

The following Python code was used:

```
df = df[['Rating', 'Review']]
df['Review'] = df['Review'].str.lower()
df.reset_index(drop=True, inplace=True)
print(df.head())
print("Dataset shape:", df.shape)
```

Output:

	Rating	Review
0	1	cool kid club snotty nosed rich kids run aroun...
1	1	donuts are really good if they have any when y...
2	1	awful not sure it classifies as food burrito c...
3	1	absolutely disgusting i had enchiladas and a t...
4	1	they served us stale rice average main dishes ...

Dataset shape: (10000, 2)

5.15 Removing URLs from Reviews

URLs in text data do not contribute to sentiment or rating analysis and can introduce noise. To address this, the following steps were applied:

- **Identify URLs:** Patterns starting with `http`, `https`, or `www` are detected using regular expressions.
- **Remove URLs:** All detected URLs are replaced with an empty string, effectively removing them from the review text.

This step ensures that the text is clean and focused only on meaningful content for further processing.

The following Python code was used:

```
def remove_url(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub('', text) # Replace URLs with empty string
df['Review'] = df['Review'].apply(remove_url)
df['Review'].head()
```

Output:

0	cool kid club snotty nosed rich kids run aroun...
1	donuts are really good if they have any when y...
2	awful not sure it classifies as food burrito c...
3	absolutely disgusting i had enchiladas and a t...
4	they served us stale rice average main dishes ...

Name: Review, dtype: object

5.16 Removing HTML Tags from Reviews

HTML tags in review text do not carry semantic meaning for analysis and can interfere with text processing. To address this, the following steps were applied:

- **Identify HTML Tags:** Patterns enclosed within `<` `>` are detected using regular expressions.
- **Remove HTML Tags:** All HTML tags are removed from the text, leaving only the readable content.

This step ensures that the dataset contains clean, human-readable text suitable for further preprocessing and feature extraction.

The following Python code was used:

```
def remove_html_tags(text):
    pattern = re.compile('<.*?>')
    return pattern.sub(r'', text)
df['Review'] = df['Review'].apply(remove_html_tags)
df['Review'].head()
```

Output:

```
0    second to worst dining experience of all time ...
1    my wife and i live around the corner hadnt eat...
2    awful not sure it classifies as food burrito c...
3    absolutely disgusting i had enchiladas and a t...
4    they served us stale rice average main dishes ...
Name: Review, dtype: object
```

5.17 Removing Punctuation and Special Characters

Punctuation and special characters in review text can introduce noise and affect text analysis. The following steps were applied:

- **Identify Punctuation:** A predefined list of punctuation symbols is used.
- **Remove Punctuation:** All punctuation characters are removed from the text.
- **Normalize Spaces:** Extra spaces created after removal are reduced to a single space, ensuring clean and readable text.

This step ensures that the dataset is clean, standardized, and ready for natural language processing.

The following Python code was used:

```
import string
punctuation_list = string.punctuation
print(f'The punctuations are {punctuation_list}')
def clean_text_punct(text):
    for char in punctuation_list:
        text = text.replace(char, ' ')
    text = ' '.join(text.split())
    return text
df['Review'] = df['Review'].apply(clean_text_punct)
df['Review'].head()
```

Output:

```
The punctuations are !"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~

0    second to worst dining experience of all time ...
1    my wife and i live around the corner hadnt eat...
2    awful not sure it classifies as food burrito c...
3    absolutely disgusting i had enchiladas and a t...
4    they served us stale rice average main dishes ...
Name: Review, dtype: object
```

5.18 Showing Stopwords in Reviews

Stopwords are common words (like “is”, “the”, “and”) that usually do not carry significant meaning for analysis. To identify them, the following steps were applied:

- **Tokenization:** Each review is tokenized using spaCy, splitting text into individual words.
- **Identify Stopwords:** Words recognized as stopwords by spaCy are extracted from each review.
- **Store Stopwords:** The identified stopwords are stored in a separate column to understand which common words are present in each review.

This step helps in analyzing and later removing non-informative words to improve feature extraction and model performance.

The following Python code was used:

```
import spacy
nlp = spacy.load("en_core_web_sm")
def show_stopwords_in_text(text):
    doc = nlp(text)
    present_stopwords = [token.text for token in doc if token.is_stop]
    return present_stopwords
df['Stopwords_in_Review'] = df['Review'].apply(show_stopwords_in_text)
df[['Review', 'Stopwords_in_Review']].head()
```

Output:

	Review	Stopwords_in_Review
0	second to worst dining experience of all time ...	[to, of, all, our, was, a, which, is, but, it,...
1	my wife and i live around the corner hadnt eat...	[my, and, i, around, the, had, here, in, a, fe...
2	awful not sure it classifies as food burrito c...	[not, it, as, of, a, and, no, they, you, anyth...
3	absolutely disgusting i had enchiladas and a t...	[i, had, and, a, just, to, give, them, a, this...
4	they served us stale rice average main dishes ...	[they, us, not, as, not, enough, in, the]

5.19 Stopword Analysis

To understand the prevalence of non-informative words (stopwords) in the review text, which can influence model performance if not handled properly.

Implementation

- For each review, the number of stopwords was calculated using the preprocessed list of stopwords obtained via **spaCy**.
- A new column, `Stopword_Count`, was created to store the count of stopwords per review.
- The total number of stopwords across the entire dataset was computed to quantify their overall presence.

Python Implementation

```
# Count stopwords in each review
df['Stopword_Count'] = df['Stopwords_in_Review'].apply(len)

# Display sample reviews with stopwords counts
df[['Review', 'Stopwords_in_Review', 'Stopword_Count']].head()

# Total stopwords in dataset
total_stopwords = df['Stopword_Count'].sum()
print("Total stopwords in all reviews:", total_stopwords)
```

Interpretation

- Counting stopwords helps evaluate the density of commonly used words that do not contribute significant semantic meaning.
- This analysis supports the decision to remove stopwords during preprocessing, reducing noise and improving the efficiency of **TF-IDF feature extraction**.
- Understanding stopwords distribution can also provide insights into the writing style and verbosity of reviews.

The total stopwords indicate the proportion of text dominated by common words versus meaningful keywords, guiding feature engineering decisions.

Output:

	Review	Stopwords_in_Review	Stopword_Count
0	cool kid club snotty nosed rich kids run aroun...	[around, than, if, you]	4
1	donuts are really good if they have any when y...	[are, really, if, they, have, any, when, you, ...	31
2	awful not sure it classifies as food burrito c...	[not, it, as, of, a, and, no, they, you, anyth...	38
3	absolutely disgusting i had enchiladas and a t...	[i, had, and, a, just, to, give, them, a, this...	33
4	they served us stale rice average main dishes ...	[they, us, not, as, not, enough, in, the]	8

Total stopwords in all reviews: 699800

5.20 Removing Stopwords from Reviews

Stopwords are removed from the review text to reduce noise and focus on meaningful words that contribute to sentiment and rating analysis. The following steps were applied:

- **Tokenization:** Each review is tokenized using spaCy to separate individual words.
- **Filter Stopwords:** Words identified as stopwords by spaCy are excluded from the text.
- **Reconstruct Text:** The remaining words are joined back into a clean, processed review.

This step ensures that the text contains only informative words, improving the effectiveness of feature extraction and model training.

The following Python code was used:

```
def remove_stopwords(text):
    doc = nlp(text)
    filtered_text = " ".join([token.text for token in doc if not token.
                              is_stop])
    return filtered_text
df['Review'] = df['Review'].apply(remove_stopwords)
df['Review'].head()
```

Output:

```
0    second worst dining experience time waitress l...
1    wife live corner nt eaten months got food mong...
2    awful sure classifies food burrito consists fl...
3    absolutely disgusting enchiladas taco chance w...
4    served stale rice average main dishes flavorfu...
Name: Review, dtype: object
```

Why spaCy over NLTK

The **spaCy** library was chosen over NLTK for text preprocessing due to several advantages:

- **Speed and Efficiency:** spaCy is faster and optimized for large-scale text processing.
- **Integrated NLP Pipeline:** spaCy provides tokenization, part-of-speech tagging, stopword detection, and lemmatization in a single pipeline.
- **Better Accuracy:** spaCy's linguistic models are highly accurate for modern NLP tasks.
- **Ease of Use:** spaCy offers a more intuitive and consistent API for preprocessing compared to NLTK.

Using spaCy ensures high-performance preprocessing and a streamlined workflow for feature extraction and model training.

5.21 Applying Lemmatization

Lemmatization converts words to their base or root form, helping to standardize the text and reduce redundancy in feature representation. The following steps were applied:

- **Generate Lemma List:** Each review is tokenized using spaCy, and the lemma for each word is extracted and stored as a list.
- **Create Joined Text:** The lemmatized words are joined back into a single string to form a normalized review text.
- **Dual Representation:** Both list format and joined text format are stored to facilitate further analysis and feature extraction.

This step ensures that the dataset is clean, normalized, and ready for vectorization, improving the performance of machine learning models.

The following Python code was used:


```
def lemmatize_text(text):
    doc = nlp(text)
    lemma_list = [token.lemma_ for token in doc] # list format
    lemma_joined = " ".join(lemma_list) # joined format
    return lemma_list, lemma_joined
df[['Review_List', 'REVIEW']] = df['Review'].apply(lambda x: pd.Series(
    lemmatize_text(x)))
print("List format:\n", df['Review_List'].head())
print("\nJoined format:\n", df['REVIEW'].head())
```

Output:

```
List format:
0    [second, bad, dining, experience, time, waitre...
1    [wife, live, corner, not, eat, month, get, foo...
2    [awful, sure, classifie, food, burrito, consis...
3    [absolutely, disgusting, enchiladas, taco, cha...
4    [serve, stale, rice, average, main, dish, flav...
Name: Review_List, dtype: object

Joined format:
0    second bad dining experience time waitress lit...
1    wife live corner not eat month get food mongol...
2    awful sure classifie food burrito consist flou...
3    absolutely disgusting enchiladas taco chance b...
4    serve stale rice average main dish flavorful m...
Name: REVIEW, dtype: object
```

Why Lemmatization is Preferred Over Stemming

- **Preserves Meaning:** Lemmatization converts words to their base or dictionary form (e.g., “better” → “good”), maintaining semantic meaning. Stemming may produce non-words (e.g., “studies” → “studi”), which can reduce interpretability.
- **Improves Accuracy:** Machine learning models benefit from normalized words that retain context, leading to better feature representation and predictive performance.
- **Handles Irregular Forms:** Lemmatization considers part-of-speech and context, making it more precise for text analysis compared to rule-based stemming.

Lemmatization is preferred for tasks requiring accurate linguistic processing, while stemming is faster but less precise. In this project, spaCy’s lemmatization ensures clean, meaningful, and consistent text for model training.

5.22 Filtering Reviews by Word Count

Reviews that are too short or excessively long can reduce model effectiveness and introduce noise. The following steps were applied:

- **Calculate Word Count:** Each review is tokenized, and the number of words is counted to assess its length.
- **Apply Length Filter:** Reviews with fewer than 3 words or more than 300 words are removed.

- **Resulting Dataset:** The filtered dataset is smaller, cleaner, and better suited for training machine learning models.

This step improves data quality, ensures relevant textual content, and enhances model performance.

The following Python code was used:

```
df['Word_Count'] = df['REVIEW'].apply(lambda x: len(x.split()))
df = df[(df['Word_Count'] >= 3) & (df['Word_Count'] <= 250)]
print("Original dataset shape:", df.shape)
print("Filtered dataset shape:", df.shape)
df[['REVIEW', 'Word_Count']].head()
```

Output:

```
Original dataset shape: (9868, 6)
Filtered dataset shape: (9868, 6)
```

	REVIEW	Word_Count
0	second bad dining experience time waitress lit...	100
1	wife live corner not eat month get food mongol...	61
2	awful sure classifie food burrito consist flou...	39
3	absolutely disgusting enchiladas taco chance b...	21
4	serve stale rice average main dish flavorful m...	11

5.23 Calculating Word Count

Word count provides insight into the length and content richness of each review, which helps in filtering and analysis. The following steps were applied:

- **Split and Count Words:** Each review is split into individual words, and the total number of words is calculated.
- **Store Word Count:** A new column, `Word_Count`, is created to store the word count for each review.

This allows for identifying very short or excessively long reviews, facilitating better preprocessing and ensuring the dataset contains informative textual content.

The following Python code was used:

```
df_balanced.loc[:, 'Word_Count'] = df_balanced['REVIEW'].str.split().str.len()
df_balanced[['REVIEW', 'Word_Count']].head()
```

Output:

	REVIEW	Word_Count
0	zipp rock place tv screen viewable angle atmos...	33
1	probably overrate place phoenix area pizza tas...	38
2	bad restaurant term hygiene friend food poison...	82
3	sis recommend place weekly dinner share large ...	43
4	ve probably pass place thousand time stop toda...	82

6. Data Visualization

6.1 Bar Plot: Review Count per Rating

A bar plot is a graphical representation that uses rectangular bars to show the frequency or count of values in a categorical variable. In this case, it displays the number of reviews for each rating.

Purpose:

- Visually represent the distribution of review ratings in the dataset.
- Helps quickly identify which ratings are more common and detect class imbalance.

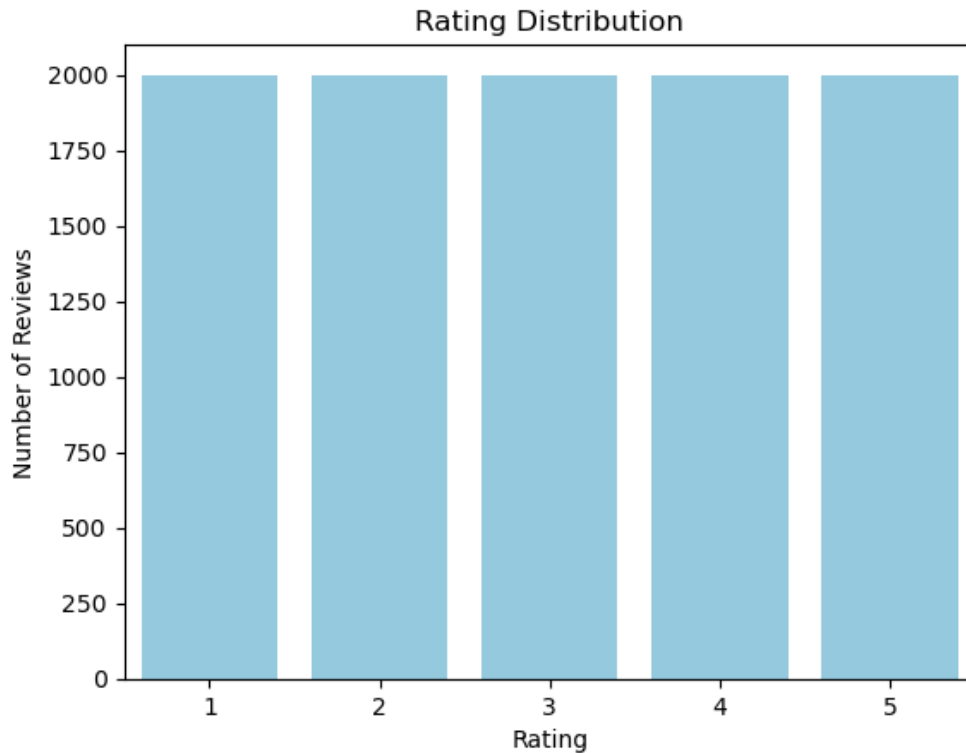
Axes and Labels:

- **X-axis:** Rating values (1–5 stars).
- **Y-axis:** Count of reviews for each rating.

Importance: Understanding rating distribution is crucial for model training, as imbalanced data can lead to biased predictions. A bar plot provides an intuitive and easy-to-interpret summary of the dataset. This visualization informs preprocessing or sampling strategies to address class imbalance.

The following Python code was used to generate the bar plot:

```
sns.countplot(x='Rating', data=df, color='skyblue')
plt.title('Review Count per Rating')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```



Figure_1: Balanced Dataset Review Count per Rating Visualization

6.2 Pie Chart: Rating Distribution

A pie chart is a circular chart divided into slices to illustrate numerical proportions. Here, it visualizes the proportion of reviews for each rating.

Purpose:

- Show the percentage share of each rating (1–5 stars) in the dataset.
- Complements the bar plot by providing a proportional view of the dataset.

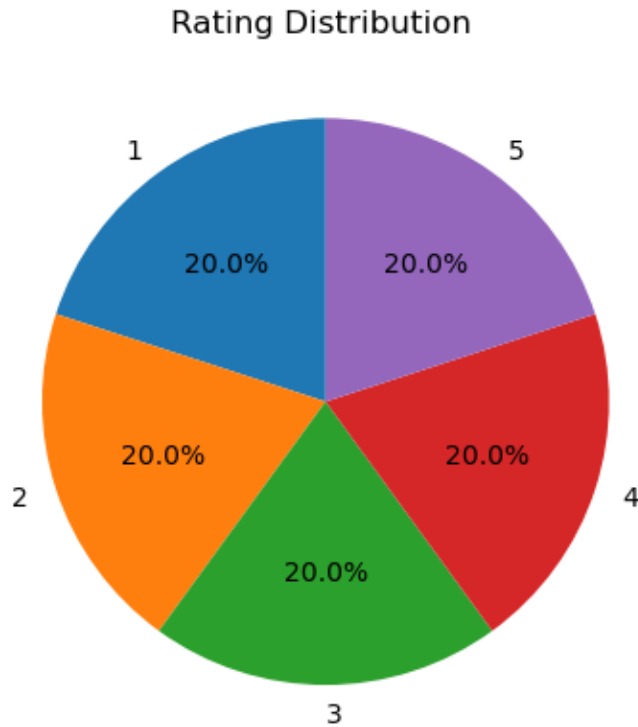
Implementation:

- `value_counts()` is used to count the number of reviews per rating.
- Labels represent the rating values, and slices show their relative percentages.

Importance: Pie charts make it easy to identify dominant ratings and detect imbalance visually. This visualization provides a clear overview of class distribution, supporting decisions for balancing and sampling strategies.

The following Python code was used to generate the pie chart:

```
rating_counts = df['Rating'].value_counts().sort_index()
categories = rating_counts.index
values = rating_counts.values
plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=90)
plt.title("Rating Distribution ")
plt.show()
```



Figure_2: Balanced Dataset Rating Distribution Visualization

6.3 Histogram: Word Count Distribution

A histogram is a graphical representation that organizes data into bins to show the frequency distribution of a numerical variable. Here, it displays how review lengths vary across the dataset.

Purpose:

- Visualize the distribution of word counts in reviews.
- Identify patterns such as very short or very long reviews.

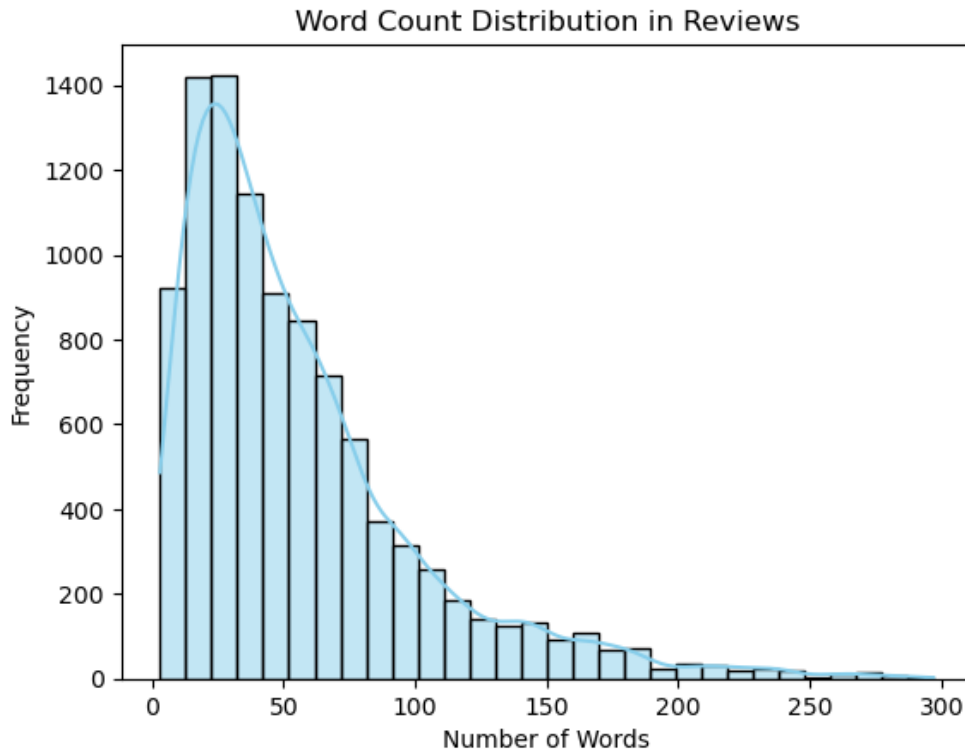
Implementation:

- `bins=30` divides the word count range into 30 intervals.
- `kde=True` adds a smooth curve to represent the probability density function.

Importance: Understanding review length distribution aids in filtering out extremely short or excessively long reviews, improving preprocessing quality. This ensures the dataset is balanced and informative for model training. The visualization provides insight into the textual richness of the reviews, supporting better decisions in data cleaning and feature extraction.

The following Python code was used to generate the histogram:

```
sns.histplot(df_balanced['Word_Count'], bins=30, kde=True, color='skyblue')
plt.title('Word Count Distribution in Reviews')
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.show()
```



Figure_3: Balanced Dataset Word Count Distribution in Review Visualization

6.4 Box Plot: Word Count Distribution by Rating

A box plot is a graphical representation that shows the distribution of a numerical variable through its quartiles, highlighting the median, spread, and outliers. Here, it illustrates how review lengths vary for each rating.

Purpose:

- Compare word count distributions across different rating categories.
- Identify patterns, such as whether higher-rated reviews tend to be longer.

Implementation:

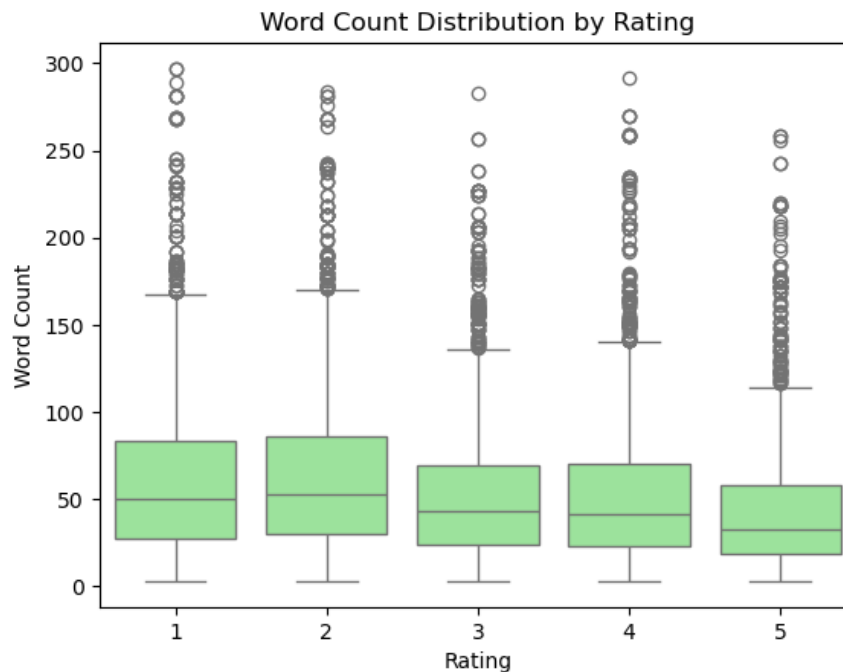
- X-axis: Rating values (1–5 stars).
- Y-axis: Word count of each review.
- The box represents the interquartile range (IQR), the line inside shows the median, and whiskers indicate variability outside the IQR.

Importance: Detects outliers and unusual review lengths that might affect model performance. Provides insight into text length trends relative to ratings, informing preprocessing and feature engineering decisions. This visualization helps ensure a better understanding of review patterns and supports more effective model training.

The following Python code was used to generate the box plot:

```
sns.boxplot(x='Rating', y='Word_Count', data=df_balanced, color='lightgreen')
plt.title('Word Count Distribution by Rating')
```

```
plt.xlabel('Rating')
plt.ylabel('Word Count')
plt.show()
```



Figure_4: Word Count Distribution by Rating Visualization

7. Sample Reviews per Rating

For the balanced dataset, up to five reviews were randomly selected from each rating category (1–5), ensuring equal representation across all classes. This approach provides a clear view of the language, sentiment, and content associated with each rating. Sampling with a fixed random seed guarantees reproducibility. By analyzing these samples, patterns in customer feedback can be identified fairly, preventing any bias toward a particular rating. The balanced selection ensures that machine learning models trained on this dataset receive uniform exposure to all categories, improving prediction reliability and performance across all rating levels.

Python Code to Extract Sample Reviews:

```
for rating, group in df_balanced_final.groupby('Rating'):
    print(f"\n--- Rating: {rating} ---\n")
    sample_reviews = df_balanced_final[df_balanced_final['Rating'] ==
                                        rating]['REVIEW'].sample(
        n=min(10, len(df_balanced_final[df_balanced_final['Rating'] ==
                                        rating])),
        random_state=42)
    for i, review in enumerate(sample_reviews, 1):
        print(f"{i}. {review}\n")
```

To gain qualitative insight into the dataset, reviews are grouped by their star ratings, and up to five random samples are shown for each rating category. This helps understand text patterns and sentiment variations across ratings. All text is shown in full to illustrate typical content, sentiment, and language patterns.

Sample Reviews

--- Rating: 1 ---

1. hopeful october scottsdale yummy tacoswow rip 10 shade buying drink place not salsa taco sauce measly price taco like vulture not care customer 2 ticket grab beg napkin hunt place sit feel watch vulture wait ready leave 45 minute pepsi get warm get cup ice bar pour take drink leave hike car security stop not let booze angry little local dump trash say sweet daughter law mad outta bring 2 year old thinking fun way spend saturday morning encourage local business nope good
2. donut worst ve buy 2 dozen donut 1099 sale bad donut life donut dry stale go couple mile buy krispy kreme krispy kreme donut fresh one worst sit shelf week supermarket
3. refuse water young teen refer pf chang olive ivy handle poorly unprofessionally pathetically pf chang not hesitate water s eat instead
4. honestly not go do gringo good friend weekend phoenix excite city offer enter do gringo sit table 21 minute waitress come table bring 2 beer come 40 minute decide check new bar obviously not welcome do gringo go bar wait 5 minute bartender picture shot friend not understand maybe 20 people 6 people work owner manager embarrass run establishment return highly recommend stay away do gringo disappoint
5. expect theme place call heart attack grill mirrors bar skimpy outfits nurse wear bar van drive section paint look like ambulance hamburger name cardiac event food bad greasy unlimited fry giant hamburger taste mixed filler kind experience plan go enjoy experience cause food head

--- Rating: 2 ---

1. thing like fez pomegranate margarita food well
2. m give restaurant 2 star typical hotel restaurant not actually sit restaurant area bar lounge service slow good night waitress ignore chit chatting table food ok d rate level chilis upscale selection appetizer base not want dinner friend breakfast morning say average usually well review restaurant yelp beg differ
3. particular restaurant set section 4 soup clearly label black kettle soup tiny little label pick read center counter bread bin lift plastic door hold open retrieve bread fascinate watch people technique bread retrieval eat tortilla chip bin end counter taco meat end cheese lettuce sour cream etc poorly plan small selection

sweet tomato pay dollar well selection

4. mill avenue farmer market bantam weight farmer market valley weigh 113 pound corner valley mill avenue farmer market let ready haggle arrive 945 8 stall produce diminutive come mind mill ave farmer market report diminutive smatter stall little produce find petite mill avenue farmer market need little padding run college girl know m say pee wee mill avenue farmer market good game today kiwanis rec center celebrate gatorade granola bar picnic area idea leave coffee return half hour later find 34 vendor honey vendor hummus cookie produce couple sandwich place flowers and place produce ill time new m hope month well turnout vendor
5. not breonna server not ask want drink decide stick appetizer not appear know water remember table food good

--- Rating: 3 ---

1. lot well use service get bad bad food not good happy hour amazing
2. travel phoenix yay chance travel sky harbor boo s decent chance ll consider travel southwest boo airline boo sky harbor boo bad southwest m huge fan southwest ill credit credit bag fly free thing genius combine favorite carrier probably favorite airport ve get recipe disaster work southwest benefit gate hard reach sw carrier banish edge universe checkin baggage claim ve get decent presence actually find human help problem yes leave strand runway 100 degree plus weather check force dash airport late arrive flight check painless operation
3. sis recommend place weekly dinner share large appetizer platter hummus baba ganoush tabouli dolmade pitas good perfect light dinner good thing fresh lemonade wow like tart ll love think overall service great waitress extremely attentive manager stop sure enjoy food flame kabob 3 star
4. thing sure leave harlow think huevos pretty good biscuit gravy leave little desire meat eat friend fair comparison amazing freshly biscuit veggie gravy biscuit recipe credit mark bittman cook decent people watch locale weekend long stand wait parking lot party inside
5. move san francisco search decent chinese food flo standby try jade palace recommendation midwestern snowbird friend comment food order order ready 15 minute complete chow fun equivalent flo ok spectacular lot oily husband mongolian beef say flo well meat taste fresher jade palace version jade palace close try hot sour soup fall usually barometer use measure chinese food figure hope san francisco restaurant relocate scottsdale

--- Rating: 4 ---

1. pra quem est mit um bom passeio universidade gigantesca e muito legal entender e absorver um pouco da cultura
2. go birthday quality insure restaurant inside season wrong restaurant classy nice upscale environment nice view phoenix area service definitely topnotch food excellence sea bass filet mignon yummy customize birthday desert visit sure talavera not forget grab drink patio live music watch bird fly mountain resort build mountain view
3. reviews extol virtue location want add experience observation time write ve phoenix year learn necessary place new town opportunity rate business objectively like mom go frys standard place hour great 5 1 location price competitive convenient vip card electronic coupon program specifically need location good pharmacy organichealth selection stock bakery deli party special occasion parking reasonable lot keptsecure s chase branch nearby good buy staple old navy wednesday s farmer market street not find fry carry thing instance depend location carry pita bread garlic naan large fage yogurt mexican version creme fraiche appreciate greatly word caution firm staff deli number insist attention not let gyp small wing pasta selection know not care job not let ruin experience remember probably make minimum wage oh reviewer comment parking accurate m sure mall complex
4. favorite breakfastbrunch spot fresh bagel different type cream cheese bagel thin cut carb not pay person take order line pay food ice coffee great deal 20 oz cup 3
5. like start restaurant review m vegetarian choice food limit consideration ve breakfast good breakfast world give 5 star write review day go yesterday sandwich salad think well little bummed instruction lack instruction staff order appear not know order pick ingredient want add delete certain sandwich salad get got not exactly include like not today star like breakfast ill mix review 4 star fair ill try m rush not avoid place breakfast want french toast waffle pancakes ill head second shot sure

--- Rating: 5 ---

1. cafe monarch restaurant experience analogy restaurant like watch concert tv place like row great immerse christopher beautiful world entertain entire time camille h mention place menu provide pick thing place quick snack bet ask christopher provide picky eater food allergy ahead time mindset go surprised meal fresh ingredient wonderful combination flavor fantastic story dish disappoint sure bring year old place enjoy think husband dining experience babysitter christopher man chef server host story teller etc course take awhile child attention span likely entire meal description amazing food eat camile hs review describe perfectly new favorite place valley not wait highly recommend experience

2. fantastic campus take short tour able lot depth exploring bring camera memorial hall
3. want student harvard visit austin hall faculty law feel legal flavour library unqiue piece gardens well park well lafayette park dc green maintain
4. dd great food friendly staff wonderful patio breakfast menu good downtown scottsdale dd great lunch early evening happy hour menu great value time food service consistently rate
5. oh goodness hank awesome ve problem landscaper finally find stick hank professional knowledgeable responsive prompt communicate thoroughly reasonable wayhe actually work trust ve gone give hope hank well expected not want tell not busy

8. Natural Language Processing (NLP)

8.1 Shuffling the Dataset

To ensure randomness and avoid order bias in model training, the balanced dataset was shuffled. This process helps distribute samples uniformly, ensuring that training and test sets are more representative of the entire dataset.

Python Implementation:

```
df_shuffled = df_balanced_final.sample(frac=1, random_state=42).
    reset_index(drop=True)
df_shuffled[['REVIEW', 'Rating']].head()
```

	REVIEW	Rating
0	not want write review place near dear heart ba...	4
1	history architecture site building able walk g...	5
2	pizzait like pizza pillow tasty savory doughy ...	3
3	decent place legit chinese food usual fast ser...	3
4	look car wash vacuum area selectione francis s...	1

Purpose:

- Prevents bias that could occur if reviews are ordered .
- Ensures randomness when splitting the dataset into training and testing sets.

Importance: Shuffling increases the generalizability of the model by reducing the risk of systematic patterns influencing learning.

8.2 Train-Test Split

To evaluate the model's performance effectively, the dataset was divided into training and testing sets. A stratified split was used to preserve the distribution of ratings across both sets, ensuring balanced representation of each class.

Purpose:

- **Training Set:** Used to build and train the machine learning model.
- **Test Set:** Reserved for evaluating model performance on unseen data.
- **Stratified Split:** Maintains the proportion of each rating class in both sets to avoid class imbalance.

Python Implementation:

```
from sklearn.model_selection import train_test_split
X = df_balanced_final['REVIEW'] # Text data
y = df_balanced_final['Rating'] # Target labels
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
print("Training set size:", len(X_train))
print("Test set size:", len(X_test))
```

```
Training set size: 8000
Test set size: 2000
```

The stratified split ensures that each rating category is represented proportionally in both the training and test sets, improving the reliability and fairness of model evaluation.

8.3 Text Preprocessing using spaCy

Before vectorizing the text, reviews were preprocessed to standardize and clean the data. The preprocessing steps include:

- **Lowercasing:** Convert all text to lowercase for uniformity.
- **Tokenization:** Split text into individual words.
- **Stopword Removal:** Remove common words that do not contribute to meaning (e.g., “the”, “is”).
- **Alphabetic Filtering:** Keep only alphabetic tokens to remove numbers and symbols.
- **Lemmatization:** Convert words to their base form to capture meaning effectively.

spaCy was preferred over NLTK because it provides faster processing, more accurate lemmatization, and integrated support for stopwords removal.

Python Implementation:

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
def spacy_preprocess(text):
    doc = nlp(text.lower()) # Lowercase and tokenize
    tokens = [token.lemma_ for token in doc if not token.is_stop and
               token.is_alpha]
    return ' '.join(tokens)
X_train_processed = X_train.apply(spacy_preprocess)
X_test_processed = X_test.apply(spacy_preprocess)
print("X_train_preprocessed:")
print(X_train_processed.head())
print("X_test_preprocessed:")
print(X_test_processed.head())
```

The resulting processed text is clean, standardized, and suitable for feature extraction and machine learning model training.

```
X_train_preprocessed:
2203    coffeemeh great horrible scondry serviceridic...
9737    impress stop x wait long reservation time frie...
8977    visit jan order thai basil signature dish cala...
8396    food middle road staff rude like start owner t...
4713    accuse training wife compensation tell leave g...
Name: REVIEW, dtype: object
X_test_preprocessed:
4220    ok m del taco chance good chicken idem um burr...
1413    couple time block live typical neighborhood ba...
2355    chip salsa star salsa food good way eat fajita...
1988                                     good bbq thank later
6214    order return chicken tender undr cook rubbery ...
Name: REVIEW, dtype: object
```

8.4 TF-IDF Vectorization

After preprocessing the text, the next step is converting textual data into numerical features that machine learning models can understand. TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used technique for this purpose. It assigns weights to each term in the corpus based on its importance in a document relative to the entire dataset.

TF-IDF Concept

TF-IDF combines two metrics: Term Frequency (TF) and Inverse Document Frequency (IDF).

- **Term Frequency (TF):** Measures how frequently a word occurs in a document.

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Inverse Document Frequency (IDF):** Measures how important a term is by reducing the weight of terms that appear in many documents (common words) and increasing the weight for rarer terms.

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

Where:

- N = Total number of documents

- n_t = Number of documents containing term t
- **TF-IDF Weight:** Combines TF and IDF to assign a weight to each term.

$$TF\text{-}IDF(t, d) = TF(t, d) \times IDF(t)$$

8.5 TF-IDF Implementation

To convert textual reviews into numerical features suitable for machine learning, Term Frequency-Inverse Document Frequency (TF-IDF) vectorization was applied. TF-IDF captures the importance of words in a document relative to the entire dataset, giving higher weight to informative words while reducing the impact of common, less meaningful terms.

Implementation Details:

- `max_features=5000`: Limits the feature space to the 5,000 most relevant terms.
- `ngram_range=(1, 2)`: Includes both unigrams (single words) and bigrams (two-word combinations) to capture context.
- `stop_words='english'`: Automatically removes common English stopwords.

Purpose:

- Converts textual reviews into numerical form suitable for machine learning model input.
- Highlights important terms while reducing noise from frequent but non-informative words.
- Preserves contextual information by including bigrams.

Outcome:

- Both the training and test sets were successfully transformed into high-dimensional numerical representations.
- These TF-IDF matrices are now ready for feature extraction, model training, and evaluation.

Python Implementation:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2),
                                   stop_words='english')
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_processed)
print("TF-IDF training set shape:", X_train_tfidf.shape)
X_test_tfidf = tfidf_vectorizer.transform(X_test_processed)
print("TF-IDF test set shape:", X_test_tfidf.shape)
print("Number of features in TF-IDF:", len(tfidf_vectorizer.get_feature_names_out()))
print("Sample features:", tfidf_vectorizer.get_feature_names_out()[:30])
```

Output:

```
TF-IDF training set shape: (8000, 5000)
TF-IDF test set shape: (2000, 5000)
```

```

Number of features in TF-IDF: 5000
Sample features: ['ability' 'able' 'absolute' 'absolutely' 'absolutely amazing'
'absolutely flavor' 'absolutely love' 'ac' 'academic' 'accent' 'accept'
'acceptable' 'access' 'accessible' 'accessory' 'accident' 'accommodate'
'accompany' 'accord' 'account' 'acknowledge' 'acoustic' 'acrid' 'acrylic'
'act' 'act like' 'action' 'active' 'activity' 'actual']

```

Why TF-IDF Over Bag of Words (BoW)

- BoW counts occurrences of words but assigns high importance to common words, which may not carry significant meaning.
- TF-IDF reduces the weight of frequent but less informative words and emphasizes rare but meaningful terms.
- By combining unigrams and bigrams, TF-IDF can capture not only individual words but also meaningful word pairs, adding contextual richness.

TF-IDF provides a more informative representation than BoW, especially for text classification tasks like predicting review ratings. It ensures that common words have lower importance, while distinctive words like “gourmet” or “seminar” carry higher significance.

8.6 Original Reviews and Ratings for Training and Test Sets

After preprocessing and shuffling, the dataset was divided into training and test sets using a stratified approach to preserve the distribution of ratings across both sets.

- **Training Set:** Contains 80% of the data and is used to train the machine learning model.
- **Test Set:** Contains 20% of the data and is used to evaluate the model’s performance on unseen reviews.

Python Implementation:

```

print("REVIEW and Rating from training set:")
train_df_sample = pd.DataFrame({'REVIEW': X_train, 'Rating': y_train}).
    reset_index(drop=True)
print(train_df_sample.head(), "\n")
print("REVIEW and Rating from test set:")
test_df_sample = pd.DataFrame({'REVIEW': X_test, 'Rating': y_test}).
    reset_index(drop=True)
print(test_df_sample.head())

```

The resulting training and test sets maintain the original rating distribution. The training set is used for model learning, while the test set allows for unbiased evaluation of model performance on unseen reviews.

Output:

REVIEW and Rating from training set:

	REVIEW	Rating
0	place bad place live live aug 2010 2011 office...	1
1	ripe old age 3 time previous review siesta cou...	4
2	okfirst let grow phx special occasion bobby ma...	3
3	want like place starter pay cash owe dollar te...	1
4	go family birthday friday night people dining ...	1

REVIEW and Rating from test set:

	REVIEW	Rating
0	place bad irs irs admit mistake arrogant worke...	1
1	new fry burb awesome old relic pretty bad ugly...	2
2	chance experience oscuro amber lager pizza ari...	3
3	good food taiwanese come mean good	5
4	go brutal hide child ve chipotle ll feel right...	1

8.7 Top TF-IDF Features per Review with Corresponding Ratings

To understand the most influential words contributing to rating predictions, **Term Frequency-Inverse Document Frequency (TF-IDF)** scores were calculated for each review in the training set. TF-IDF reflects the importance of a term in a specific document relative to its occurrence in the entire corpus.

The TF-IDF score is calculated as:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \frac{N}{1 + \text{DF}(t)}$$

Where:

- $\text{TF}(t, d)$ – Term frequency of term t in document d .
- $\text{DF}(t)$ – Number of documents containing term t .
- N – Total number of documents in the corpus.

Process:

1. Convert the sparse TF-IDF matrix to a dense format for easier manipulation.
2. Extract the top n words with the highest TF-IDF scores for each review.
3. Associate each set of top features with the corresponding review rating.

Python Implementation:

```
def top_tfidf_features_per_review(tfidf_matrix, feature_names, ratings,
    top_n=5):
    if not isinstance(tfidf_matrix, np.ndarray):
        tfidf_dense = tfidf_matrix.toarray()
    else:
        tfidf_dense = tfidf_matrix

    top_features_list = []

    for i in range(tfidf_dense.shape[0]):
```



```

row = tfidf_dense[i]
top_indices = row.argsort() [-top_n:] [::-1]
top_features = [feature_names[idx] for idx in top_indices]
top_scores = [row[idx] for idx in top_indices]
top_features_list.append({
    'Rating': ratings.iloc[i] if hasattr(ratings, 'iloc') else
        ratings[i],
    'Top_Features': top_features,
    'TF-IDF_Scores': top_scores
})

return pd.DataFrame(top_features_list)
feature_names = tfidf_vectorizer.get_feature_names_out()
top_features_df = top_tfidf_features_per_review(X_train_tfidf,
        feature_names, y_train, top_n=5)
top_features_df.head()

```

Outcome:

- **Rating** – The actual star rating of the review.
- **Top_Features** – The top n words with the highest TF-IDF scores in the review.
- **TF-IDF_Scores** – Corresponding TF-IDF scores of these words.

This approach highlights the key terms that influence the predicted ratings and aids in feature interpretation and model explainability.

Output:

	Rating	Top_Features	TF-IDF_Scores
0	1	[pay customer, customer, pay, owner come, dist...	[0.47842997695469736, 0.2303652870028023, 0.22...
1	4	[excellent, time friend, advertise, bad meal, ...	[0.2546621436179573, 0.2164445646995514, 0.205...
2	3	[dish, order, clay pot, clay, dish good]	[0.30786550848460115, 0.20181594742399292, 0.1...
3	1	[choice area, choice, wrong, fix issue, wrong ...	[0.4346146442104889, 0.3598787323201044, 0.231...
4	1	[compensation, trainer, wife, gym, training]	[0.36744444301322343, 0.3557685071338308, 0.34...

	Rating	Top_Features	TF-IDF_Scores
7995	1	[salsa, come, cilantro, taco, meat]	[0.29791307826512436, 0.2489043253494343, 0.22...
7996	2	[glass, lack, flame, ve want, miso soup]	[0.228770169236689, 0.2264513176429563, 0.1820...
7997	2	[college, bar, usually, summer heat, college kid]	[0.3648577748996445, 0.22982921323368613, 0.19...
7998	4	[decide, sunny, coffee shop, east coast, decid...	[0.3282104361251101, 0.29853927787507195, 0.29...
7999	5	[willow, shoulder, facial, spa, massage]	[0.3619608761483841, 0.2527257078372993, 0.250...

8.8 Convert TF-IDF Matrices to DataFrames with Labels

After computing TF-IDF vectors for the training data, the sparse matrix is converted into a pandas DataFrame to facilitate further analysis and model integration. Each column represents a unique term from the corpus, and each row corresponds to a single review.

Steps:

1. **Extract feature names:** Retrieve all unique words (features) identified by the TF-IDF vectorizer using `get_feature_names_out()`.
2. **Convert TF-IDF matrix to DataFrame:** Transform the sparse TF-IDF matrix into a dense format and store it as a DataFrame, where columns correspond to feature names and rows correspond to reviews.
3. **Add target labels:** Append the `Rating` column to associate each review's TF-IDF features with its corresponding star rating.

Python Implementation:

```
feature_names = tfidf_vectorizer.get_feature_names_out()
tfidf_df_train = pd.DataFrame(X_train_tfidf.toarray(), columns=
    feature_names)
tfidf_df_train['Rating'] = y_train.reset_index(drop=True)
tfidf_df_train.head()
tfidf_df_train.tail()
```

This DataFrame format simplifies analysis, visualization, and integration with machine learning models for rating prediction.

Output:

	ability	able	absolute	absolutely	absolutely amazing	absolutely flavor	absolutely love	ac	academic	accent	...	yuck	yum	yummy	yung	zero	zero star	zipp	zu	zucchini	Rating
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1

5 rows × 5001 columns

	ability	able	absolute	absolutely	absolutely amazing	absolutely flavor	absolutely love	ac	academic	accent	...	yuck	yum	yummy	yung	zero	zero star	zipp	zu	zucchini	Rating
7995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.052657	0.0	0.0	0.0	0.0	0.0	0.0	1
7996	0.0	0.0	0.0	0.0	0.0	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	2
7997	0.0	0.0	0.0	0.0	0.0	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	2
7998	0.0	0.0	0.0	0.0	0.0	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	4
7999	0.0	0.0	0.0	0.0	0.0	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	5

5 rows × 5001 columns

9. Machine Learning Model Training and Evaluation

9.1 Overview

The task of predicting review ratings from textual feedback was formulated as a multi-class classification problem. Several supervised learning algorithms were evaluated to identify the most effective model. Features were derived from preprocessed textual reviews using TF-IDF vectorization, capturing the importance of unigrams and bigrams. All experiments were conducted on a **balanced dataset**, ensuring equal representation of each rating category to avoid bias in model predictions.

9.2 Models Evaluated

The following machine learning models were evaluated:

- **Logistic Regression (LR):** A linear model for classification, achieved 77% accuracy.
- **Support Vector Machine (LinearSVC):** Utilized a linear kernel to separate classes, achieving 84% accuracy.
- **Naive Bayes (MultinomialNB):** A probabilistic model suitable for text data, achieved 68% accuracy.
- **Decision Tree (DT):** A tree-based classifier capturing non-linear relationships, achieved 83% accuracy.
- **Random Forest (RF):** An ensemble of decision trees leveraging bagging and feature randomness, achieved 86% accuracy, the highest among evaluated models.

9.3 Model Selection

Random Forest was selected as the final model due to its superior predictive performance, robustness to overfitting, and ability to handle high-dimensional feature spaces efficiently. The balanced dataset ensured consistent performance across all rating categories.

9.4 Preprocessing and Feature Extraction

Text preprocessing steps applied prior to model training included:




















- **Text Cleaning:** Removal of URLs, HTML tags, punctuation, emojis, and special characters.
- **Case Normalization:** Conversion of all text to lowercase.
- **Stopword Removal:** Elimination of non-informative words using spaCy.
- **Lemmatization:** Conversion of words to their root forms for consistent representation.
- **Word Count Filtering:** Removal of very short (<3 words) or excessively long (>300 words) reviews.
- **TF-IDF Feature Extraction:** Transformation of cleaned text into numerical vectors representing word importance.

9.5 Model Training

The balanced dataset was split into training (80%) and test (20%) sets using stratified sampling to preserve equal representation of each rating. The Random Forest classifier was trained with 1000 estimators using the TF-IDF features. This enabled the model to learn patterns connecting textual features to their corresponding ratings effectively.

Python Implementation:

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=1000, random_state=42,
                                n_jobs=-1)
rf_model.fit(X_train_tfidf, y_train)
```

RandomForestClassifier		
Parameters		
	n_estimators	1000
	criterion	'gini'
	max_depth	None
	min_samples_split	2
	min_samples_leaf	1
	min_weight_fraction_leaf	0.0
	max_features	'sqrt'
	max_leaf_nodes	None
	min_impurity_decrease	0.0
	bootstrap	True
	oob_score	False
	n_jobs	-1
	random_state	42
	verbose	0
	warm_start	False
	class_weight	None
	ccp_alpha	0.0
	max_samples	None
	monotonic_cst	None

9.6 Evaluation Metrics and Interpretation

The performance of the Random Forest classifier was evaluated on the test set using standard classification metrics:

- **Accuracy:** Achieved 87%, indicating high overall correctness of predicted ratings.
- **Precision:** High precision across all rating classes, reflecting the model's ability to avoid false positives.
- **Recall:** High recall values indicate that the model correctly identifies a majority of reviews for each rating.
- **F1-score:** Balanced F1-scores confirm robust performance across all categories, with minimal trade-off between precision and recall.
- **Confusion Matrix:** The confusion matrix revealed that most misclassifications occurred between adjacent ratings (e.g., 3 vs. 4 stars), which is reasonable due to subjective interpretation of review sentiment.

Python Implementation:

```
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
y_pred = rf_model.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

9.7 Results

Random Forest achieved 87% accuracy on the balanced dataset. Classification report:

- Class 1: Precision 0.92, Recall 0.97, F1-score 0.94
- Class 2: Precision 0.96, Recall 0.90, F1-score 0.93
- Class 3: Precision 0.91, Recall 0.86, F1-score 0.88
- Class 4: Precision 0.79, Recall 0.76, F1-score 0.77
- Class 5: Precision 0.77, Recall 0.85, F1-score 0.81

9.8 Interpretation:

The Random Forest classifier demonstrated robust performance on the balanced dataset, making it well-suited for automated review rating prediction. Its ensemble nature captures complex linguistic patterns while mitigating overfitting, ensuring reliable predictions across all rating categories.

The results confirm that the Random Forest model effectively captures linguistic patterns in reviews and maps them to appropriate ratings. The balanced dataset ensures fair representation, reducing bias towards any particular rating class and enabling reliable predictions across all categories. The combination of TF-IDF features and ensemble learning allows the model to generalize well to unseen data.

Result and Visualization

Classification Report:

	precision	recall	f1-score	support
1	0.92	0.97	0.94	400
2	0.96	0.90	0.93	400
3	0.91	0.86	0.88	400
4	0.79	0.76	0.77	400
5	0.77	0.85	0.81	400
accuracy			0.87	2000
macro avg	0.87	0.87	0.87	2000
weighted avg	0.87	0.87	0.87	2000

Confusion Matrix:

```
[[387  5  4  2  2]
 [ 18 358  9  7  8]
 [  4  4 343 24 25]
 [  9  3 18 304 66]
 [  3  3  4  50 340]]
```

Confusion matrix highlights occasional misclassification between adjacent ratings (e.g., 3 vs. 4 stars), which is expected due to subjective variations in review text.

