# Automated Review Rating System

## Intoduction:

In the digital age, customers rely heavily on online reviews before making purchasing decisions. However, many reviews lack explicit ratings or contain subjective sentiments that are difficult to interpret at scale. The goal of this project, **Automated Review Rating System**, is to build a machine learning pipeline that predicts a review's star rating (from 1 to 5) based solely on its textual content.

This system automates the process of rating reviews by:

- Cleaning and preprocessing raw review text.
- Visualizing review patterns and distributions.
- Creating a imbalanced dataset for fair model training.
- Splitting the data appropriately for training and evaluation.
- Applying vectorization techniques like **TF-IDF** to prepare the text for model input.

The resulting dataset and preprocessing steps lay the groundwork for building accurate and unbiased models capable of assigning star ratings to reviews thereby enhancing user experience and platform credibility.

## Environment Setup:

To ensure smooth execution of the project and compatibility across systems, the following environment setup is recommended:

**Python Installation**

- Use **Python 3.10+**
- **Recommended:** Install Python via **Anaconda Distribution**
  Anaconda comes preloaded with most data science libraries and Jupyter Notebook support.

**Required Python Libraries**

The following libraries are essential for this project:

- pandas
- numpy
- matplotlib
- seaborn
- scikit-learn
- spacy

Install them using:

```
pip install pandas numpy matplotlib seaborn scikit-learn spacy
```

### Additional Setup

- Download the English NLP model for spaCy:

```
python -m spacy download en_core_web_sm
```

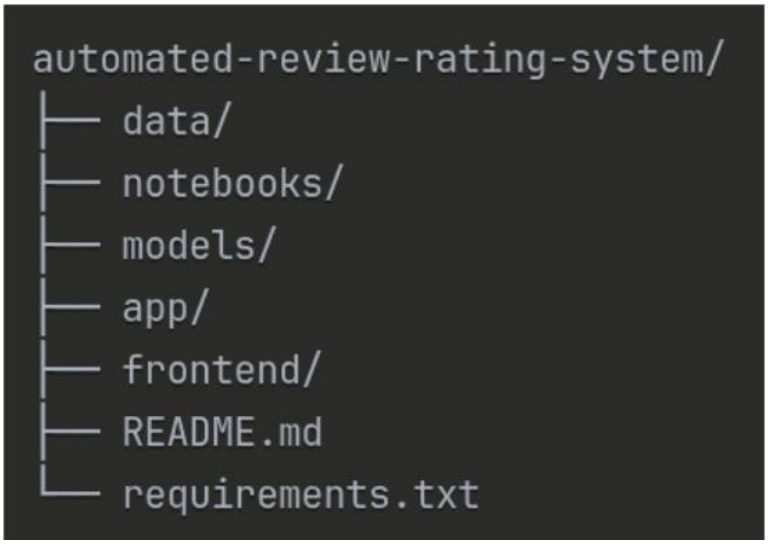
## Github Project setup:

To manage version control and enable collaboration, a GitHub repository was created for the project.

Repository Name: https://github.com/hannafarsin/automatic-review-rating-system

### Folder Structure:

The project is organized using a modular structure to ensure clarity, maintainability, and scalability:




### Initial Commit:

- Added base folder structure.

- Created an initial `README.md` with a short description of the project.
- Included `requirements.txt` for setting up the Python environment.

# 1.Data Collection:

The dataset used for this project was sourced from **Kaggle**, specifically from the **Amazon Cell Phones and Accessories Reviews** dataset.

## Dataset Overview:

- **Source**: Kaggle - Amazon Reviews: Cell Phones and Accessories
- **Dataset Link:** **https://github.com/hannafarsin/automatic-review-rating-system/blob/main/data/kaggle/reviews_output.csv**
- **Imbalanced Dataset Link** **https://github.com/hannafarsin/automatic-review-rating-system/blob/main/data/kaggle/imbalanced_reviews_dataset3.csv**
- **Format**: CSV
- **Size**: 190,000 reviews
- **Key Columns**:
  - `reviewText`: The main review written by the customer.
  - `Rating` (or `overall`): The star rating given by the user (ranging from 1 to 5).
  - Additional fields like `reviewTime`, `reviewerID`, `summary`, etc., were available but not essential for this task.

## Purpose of Data Collection:

The raw data serves as the foundation for training an automated system that can predict a review's rating purely from its text. Since the original dataset is imbalanced (i.e., more 5-star reviews than 1-star), additional balancing steps were planned as part of the preprocessing.

## Dataset Preview:

```
[4]: df = pd.read_csv('reviews_output.csv')
     df
```

[4]:

| | reviewerID | asin | reviewerName | helpful | reviewText | Rating | summary | unixReviewTime | reviewTime |
|---|---|---|---|---|---|---|---|---|---|
| 0 | A30TL5EWN6DFXT | 120401325X | christina | [0, 0] | They look good and stick good! I just don't li... | 4 | Looks Good | 1400630400 | 05 21, 2014 |
| 1 | ASY55RVNIL0UD | 120401325X | emily l. | [0, 0] | These stickers work like the review says they ... | 5 | Really great product. | 1389657600 | 01 14, 2014 |
| 2 | A2TMXE2AFO7ONB | 120401325X | Erica | [0, 0] | These are awesome and make my phone look so st... | 5 | LOVE LOVE LOVE | 1403740800 | 06 26, 2014 |
| 3 | AWJ0WZQYMYFQ4 | 120401325X | JM | [4, 4] | Item arrived in great time and was in perfect ... | 4 | Cute! | 1382313600 | 10 21, 2013 |
| 4 | ATX7CZYFXI1KW | 120401325X | patrice m rogoza | [2, 3] | awesome! stays on, and looks great. can be use... | 5 | leopard home button sticker for iphone 4s | 1359849600 | 02 3, 2013 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 194434 | A1YMNTFLNDYQ1F | B00LORXVUE | eyeused2loveher | [0, 0] | Works great just like my original one. I reall... | 5 | This works just perfect! | 1405900800 | 07 21, 2014 |
| 194435 | A15TX8B2L8B20S | B00LORXVUE | Jon Davidson | [0, 0] | Great product. Great packaging. High quality a... | 5 | Great replacement cable. Apple certified | 1405900800 | 07 21, 2014 |
| 194436 | A3JI7QRZO1QG8X | B00LORXVUE | Joyce M. Davidson | [0, 0] | This is a great cable, just as good as the mor... | 5 | Real quality | 1405900800 | 07 21, 2014 |
| 194437 | A1NHB2VC68YQNM | B00LORXVUE | Nurse Farrugia | [0, 0] | I really like it becasue it works well with my... | 5 | I really like it becasue it works well with my... | 1405814400 | 07 20, 2014 |
| 194438 | A1AG6U022WHXBF | B00LORXVUE | Trisha Crocker | [0, 0] | product as described, I have wasted a lot of m... | 5 | I have wasted a lot of money on cords | 1405900800 | 07 21, 2014 |

194439 rows × 9 columns

# 2. Data Preprocessing:

## 2.1 Initial Data Inspection

Before applying text preprocessing techniques, the dataset was examined for structural issues:

**Missing (null) values**:

Columns like `reviewText`, `Rating` were checked.

*df.isnull().any()*

```
reviewerID        False
asin              False
reviewerName       True
helpful           False
reviewText         True
Rating            False
summary            True
unixReviewTime    False
reviewTime        False
dtype: bool
```

*df.isnull().sum()*

```
reviewerID           0
asin                 0
reviewerName      3525
```

```
helpful              0
reviewText          99
Rating               0
summary              1
unixReviewTime       0
reviewTime           0
dtype: int64
```

`reviewText` or `Rating` were removed.

```
df_no_mv=df.dropna(subset=['reviewerName', 'reviewText', 'summary'], axis=0)
df_no_mv.isnull().sum()
```

```
reviewerID        0
asin              0
reviewerName      0
helpful           0
reviewText        0
Rating            0
summary           0
unixReviewTime    0
reviewTime        0
dtype: int64
```

**Duplicate entries**:

Exact duplicates were identified and dropped to avoid model bias.

```
df_no_mv.duplicated().sum()
```

```
0
```

**Remove Conflicting Reviews(Same text,Different Ratings):**

```
# Check how many reviews have conflicting ratings
conflict_counts = (df_no_mv.groupby('reviewText')['Rating'].nunique()
    .reset_index().rename(columns={'Rating': 'unique_ratings'}))
# Get texts that have more than 1 unique rating
conflicting_reviews = conflict_counts[conflict_counts['unique_ratings'] > 1]['reviewText']
# Filter original df for only conflicting reviews
conflict_df = df[df['reviewText'].isin(conflicting_reviews)]
# Group by reviewText to see all associated ratings
conflict_summary = conflict_df.groupby('reviewText')['Rating'].unique().reset_index()
#  Show a few rows
conflict_summary.head(10)
```

:

| | reviewText | Rating |
|---|---|---|
| 0 | #NAME? | [4, 5, 3, 2] |
| 1 | GOOD | [5, 3] |
| 2 | Good | [4, 5] |
| 3 | Good case | [4, 5] |
| 4 | Good product | [4, 5] |
| 5 | I bought the case for the wrong note. But I ha... | [4, 3] |
| 6 | I like it | [4, 5] |
| 7 | I love it | [4, 5] |
| 8 | Like it | [4, 5] |
| 9 | Love it | [1, 5, 4] |

## 2.2 Text Cleaning and Normalization:

In this step, the raw review text was cleaned and standardized to reduce noise and prepare it for further processing.

**# 1. Pre-split cleaning: punctuation, emojis, spaces**

```
def basic_clean(text):
    text = str(text).lower()
    text = re.sub(r'[' + string.punctuation + ']', '', text)         # remove punctuation
    text = re.sub(r'[^\x00-\x7F]+', '', text)                    # remove emojis / non-ASCII
    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'[^a-zA-Z0-9\s]', '', text)                     # remove special characters
    text = re.sub(r'\s+', ' ', text).strip()                     # remove extra spaces, tabs, etc.
    return text
df_cleaned['reviewText'] = df_cleaned['reviewText'].apply(basic_clean)
```

Cleaning Steps Applied:

### 1. **Lowercasing**

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

- All text was converted to lowercase to maintain consistency.

- Example:
  `"This PHONE is Great!"` → `"this phone is great"`

## 2. **Removing URLs and HTML Tags**

```
text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)  # remove URLs
text = re.sub(r'<.*?>', '', text)                                        # remove HTML tags
```

- Any embedded links or HTML elements were removed using regular expressions.
- Example:
  `"Check it out: <a href='...'>Link</a> "` → `"check it out"`

## 3. Removing Emojis and Special Characters

```
text = re.sub(r'[^\x00-\x7F]+', '', text)                    # remove emojis / non-ASCII
text = re.sub(r'[^a-zA-Z0-9\s]', '', text)                   # remove special characters
```

- Emojis, symbols, and non-ASCII characters were stripped from the text.
- Example:
  `"Great phone 😍🌟!"` → `"great phone"`

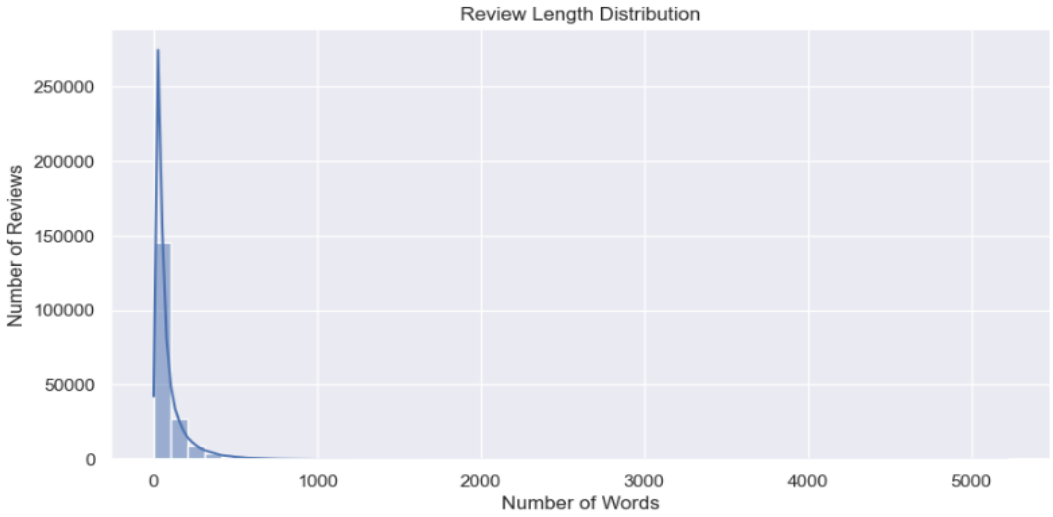
## 4. Removing Punctuation and Numbers

```
text = re.sub(r'[' + string.punctuation + ']', '', text)          # remove punctuation
```

- Punctuation marks (e.g., `! ? , .`) and digits were removed to simplify the text.
- Example:
  `"Rated 10/10!!!"` → `"rated"`

## 5. **Review Length Filtering**

- Very short reviews (fewer than 3 words) and excessively long ones (e.g., over 350 words) were removed.
- This helped eliminate outliers and non-informative entries.




## Justification for Review Length Filtering

The histogram above illustrates the **distribution of review lengths** based on word count. It shows a **heavily right-skewed distribution**, where:

- The **majority of reviews contain fewer than 100 words**, with a sharp peak under 50 words.
- A **long tail of outliers** includes reviews with extremely high word counts, some exceeding **5000 words**.

To improve data quality and processing efficiency:

- **Very short reviews** (less than 3 words) were removed, as they offer limited semantic information.
- **Excessively long reviews** (e.g., over 350–500 words) were filtered out to eliminate outliers that may introduce noise or slow down training.

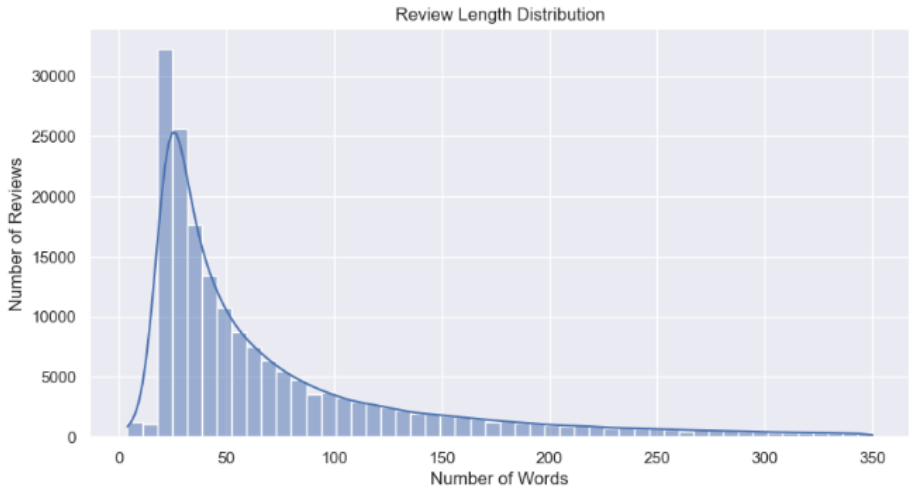
This step ensures a more **uniform distribution**, making the dataset more suitable for vectorization and model training.

## After Filtering:

- Very short reviews (fewer than 3 words) and excessively long ones (e.g., over 350 words) were removed.
- This helped eliminate outliers and non-informative entries

```
df_filtered = df_cleaned[
    (df_cleaned['review_length'] > 3) &
    (df_cleaned['review_length'] <= 350)
]
print("Before filtering:", len(df_cleaned))
print("After filtering:", len(df_filtered))
```

```
Before filtering: 190708
After filtering: 182942
```




After applying the text cleaning and length-based filtering:

- Reviews with **fewer than 3 words** and those with **excessive lengths (over 350 words)** were removed.
- The remaining dataset shows a **more uniform and meaningful distribution** of review lengths, as visualized in the updated histogram above.
- Most reviews now fall between **10 to 150 words**, providing enough context while avoiding noise from overly short or verbose entries.

# 3. Creating a Imbalanced Dataset

### Why Imbalance the Dataset?

In real-world applications such as online review systems, customer feedback often follows an **imbalanced rating distribution**. For instance, most users tend to leave 4-star or 5-star reviews when they are satisfied, while only a small fraction leave 1-star or 2-star reviews when dissatisfied.

Building and evaluating models **only on balanced datasets** can lead to overly optimistic or misleading performance metrics. In contrast, using an **imbalanced dataset during evaluation** helps to:

- **Simulate real-world conditions**
- Reveal how the model handles rare classes (like 1-star ratings)
- Uncover biases toward majority classes (e.g., 5-star reviews)
- Validate the robustness of models trained on balanced data

## The Challenge

In imbalanced datasets:

- Classifiers tend to **favor the majority class**
- Metrics like **accuracy** become less reliable (e.g., always predicting "5-star" gives high accuracy)
- **Minority classes are underrepresented**, yet are often the most important (e.g., critical feedback)

Rating Distribution across 1 through 5 stars are from te orginal Dataset:

```
df_filtered['Rating'].value_counts()
```

```
Rating
5    102530
4     37042
3     20171
1     12727
2     10472
Name: count, dtype: int64
```

As shown, over **60% of the reviews** are either 4 or 5 stars, while the 1- and 2-star reviews together make up **less than 15%**. This imbalance is typical in real-world data but can lead to models that underperform on minority classes (e.g., 1- and 2-star reviews).To simulate and analyze the impact of a controlled imbalance, we constructed a **custom imbalanced dataset** consisting of **10,000 reviews**, following the distribution:

## Imalancing Strategy:

To simulate real-world bias and evaluate model robustness under class imbalance, we constructed a **custom imbalanced dataset with exactly 10,000 reviews**, using the following rating distribution:

| Rating | Target % | Sample Count |
|---|---|---|
| ☆ 1 | 10% | 1,000 |
| ☆ 2 | 15% | 1,500 |
| ☆ 3 | 25% | 2,500 |
| ☆ 4 | 30% | 3,000 |

| Rating | Target % | Sample Count |
|---|---|---|
| ☆ 5 | 20% | 2,000 |
| | | **10,000** |

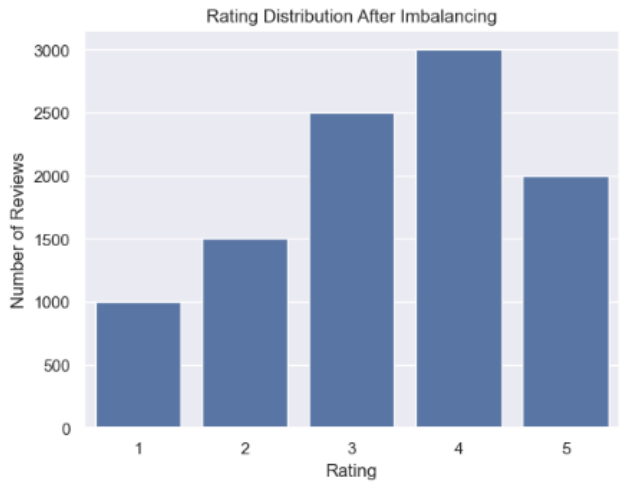
```
# Define how many samples to take for each rating
target_counts = {
    1: 1000,
    2: 1500,
    3: 2500,
    4: 3000,
    5: 2000
}
df_imbalanced_scaled = pd.concat([
    df_remaining[df_remaining['Rating'] == rating].sample(n=count, random_state=42)
    for rating, count in target_counts.items()
]).sample(frac=1, random_state=42).reset_index(drop=True)
print(df_imbalanced_scaled['Rating'].value_counts())
```

This resulted in a Im**balanced dataset of 10,000 reviews** with equal representation from each class.

**Important Note**:
To ensure independence from the model's training phase, the samples used to create this imbalanced dataset were taken **only from the portion of the dataset not used in the balanced training set**. This avoids data leakage and ensures a clean, fair evaluation of the model's performance on previously unseen examples.

```
used_balanced = df_balanced.index
df_remaining = df_filtered.drop(index=used_balanced, errors='ignore')
```

## Imbalanced Rating Distribution




The bar chart clearly illustrates the **custom imbalanced distribution** of product review ratings:

- **4-star reviews** are the most common, making up **30%** of the dataset (3,000 samples).
- **3-star reviews** contribute **25%** (2,500 samples), followed closely by **5-star reviews** at **20%** (2,000 samples).
- **Lower ratings** (1-star and 2-star) make up a smaller portion of the dataset—**10% and 15%**, respectively.

This imbalance reflects a **real-world scenario** where users are more likely to leave moderate-to-high ratings than low ratings

## Display the statistics of the word count after Imbalancing:

```
df['word_count'] = df['reviewText'].apply(lambda x: len(str(x).split()))
#display the statistics of the word count
min_word_counts = df.groupby('Rating')['word_count'].describe()
print(min_word_counts)
```

```
        count        mean         std  min   25%   50%     75%     max
Rating
1      1000.0   62.700000   54.246296  5.0  27.0  43.0   76.00   344.0
2      1500.0   78.630667   68.952185  4.0  31.0  51.0   99.25   347.0
3      2500.0   74.358800   65.506507  4.0  29.0  49.0   96.00   350.0
4      3000.0   81.103000   70.985013  5.0  29.0  53.0  109.00   350.0
5      2000.0   68.890500   65.665786  4.0  26.0  42.0   84.00   349.0
```

**Display 3 full sample reviews per rating:**

```
for rating in sorted(df['Rating'].unique()):
    print(f"\n☆ Rating {rating}")
    reviews = df[df['Rating'] == rating].tail(3)
    for _, row in reviews.iterrows():
        print(f"- {row['reviewText']}")
```

```
☆ Rating 1

- leather wallet flip caseif you read the description its fake leather thats somew
hat misleading to begin with i just received this item and its not even fake leath
erits rubber rubber is not leather nor fake leatherthe speaker slot cutout doesnt
actually line up with the speaker at allall that aside its a decent case not great
but decentupdate ive lowered this to one star as it cant even be removed ive been
fighting with this case for a half hour and managed to scratch the phone crack the
case but not budge it in the slightest impressive in its own way but not good


- i used this device for a week and it failedi am unsure if this will work with an
iphone 5 some features do not work like volume and telephoneluckily sending it bac
k for a refund was easythe fit on the head is a bit strange but manageable had the
headset worked


- super thin film is impossible to apply without many bubblesive applied window fi
lm that was easier than this stuff so im not some noob that doesnt know how to app
ly these things ive never had a problem with screen protectors in the past because
they were thick enough to not act like thick cellophanei ended up throwing them ou
t because they were totally worthlessspend the money and buy something of far bett
er quality


☆ Rating 2

- this wallet case did not fit my phone it was too small i would not recommend it
to a friend because once you get it youre not going to want to send it back becaus
e of the cost


- im happy but also a bit disappointed because the advertisment said it was new wh
en i opened the envelope it was a usb cord without the package so i felt like i wa
s duked but other than that its ok because it works and thats a good thing


- this item is a great price but i did not get what is pictured instead they sent
me another iphone 4 case i can use it but it wasnt the one i had ordered


☆ Rating 3

- i received this watch yesterday so far its ok but it has some issues the heard r
ate monitor takes a couple of times to work the notifications dont display very lo
ng and theres no way to adjust the time display setting also since samsung using t
heir tizen os theres not that many third party apps but samsung released their sdk
```

so hopefully theyre be some out soon i wish there were more clock faces but im sur e people will make new ones in the near future

- the product is advertised as 34scratch resistant34 however within a few days the screen protector cover had scratches all over it these are difficult to apply no m atter the brand you choose sure they work but once i bought an otterbox that comes with a screen protector built in no more worries no more dealing with lining up sc reen protectors or dealing with bubbles between the screen and protector with how expensive these things can be and how many you can go through its not worth it to me to purchase again

- better that the official but the volume rocker and the power button at the month get peel off nice customer service replacing the bumper but still having the same issue

☆ Rating 4

- the belkin shield blooms case cover is light rounded and with soft edges relativ ely soft flexible and thin exactly as much as needed for protection below 116 inch fitting perfectly not slippery does not cover anything requiring direct access inc luding controls and dock connector allowing charging while in case and has a very nice surface that will mask marks of use the pattern is flush with the background though it makes a texture that can be felt the surface is not sticky and does not attract grime though it can be easily washed in the unlikely event it is inferior to the covers which instead of some cutouts have very functional protrusions for t he buttons allowing to feel and press them as without the cover such as the simila rly priced but boring infinite products plasma tpu case and belkin grip neon glo c ase

- everyone really liked the wallet one thing missing a place to put the phone on h is beltso the phone is easier to get to

- not a bad little cover they are easy to apply and hold well my only issue is tha t there is not a cut out for the front camera there for if you take a picure it is very fuzzy and unusable i could just cut out a square but i am too lazyi will orde r them again if neededshipping was fast

☆ Rating 5

- i dont have to worry about extra cable from regular headphones i can use them on the phone or to listen to music the battery lasts a relatively long time theyre su per light provide stereo sound since it has two earbuds and i use them most while working out when in use they do get some attention their design is very unique but thats ok by me im proud of them bought them in white to match my white iphone thou gh i have seen the black which maybe dont draw so much attention

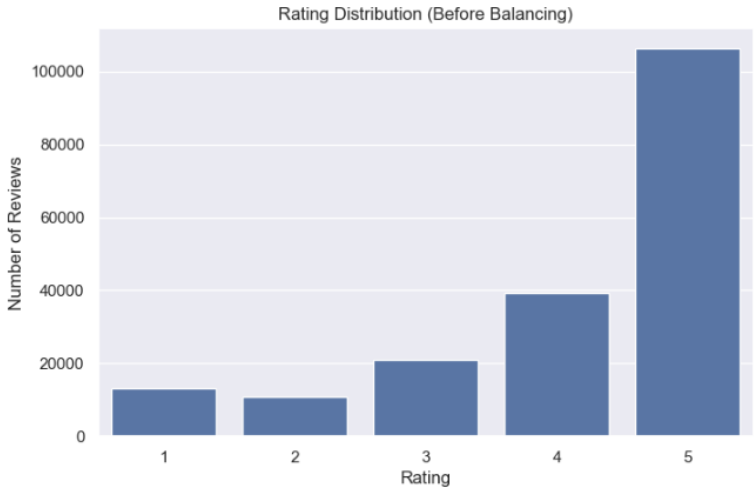
- ive bought quite a few cases for my phone but this is the only one that seems to be worth the money i ended up giving it to my nephew because he asked for it and i ts still like new

```
- these items took a little time to deliver but they are nice bumper cases for the iphone 5i would assume these would work with the iphone 5s as wellthe only issue with bumper cases is that there is no plastic backing if you want something like that i would recommend the griffin reveal cases
```

# 4. Data Visualization

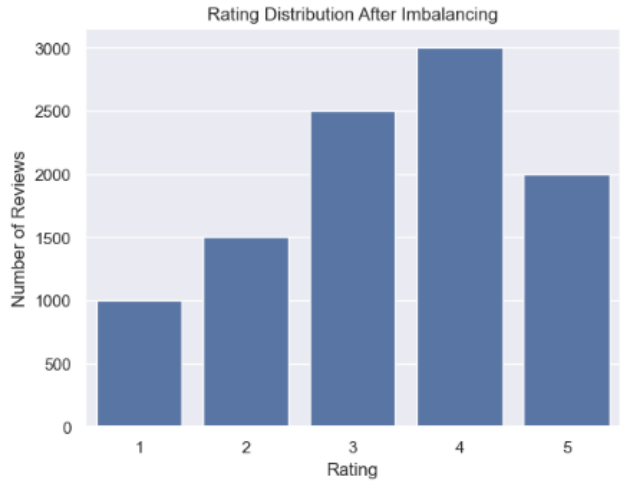
To better understand the characteristics of the reviews and guide preprocessing decisions, several visualizations were created. These plots offer insights into rating distribution, review length, and review content.

## 4.1 Review Rating Distribution (Before Imbalancing)




**Insight:** The original dataset was highly imbalanced, with over 100,000 reviews rated 5 stars, while 1- and 2-star reviews were much less frequent. This confirmed the need for dataset balancing before training.

## 4.2 Review Rating Distribution (After Imbalancing)

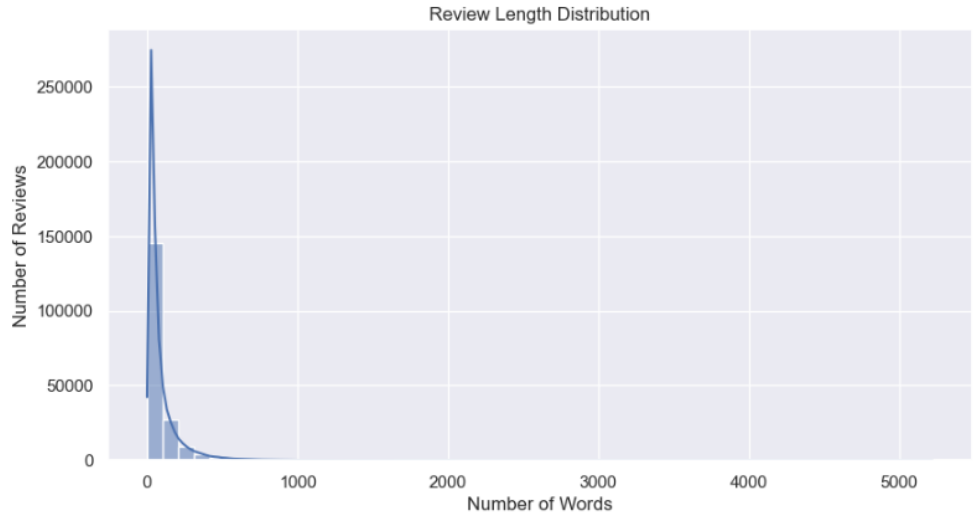


The bar chart clearly illustrates the **custom imbalanced distribution** of product review ratings:

- **4-star reviews** are the most common, making up **30%** of the dataset (3,000 samples).
- **3-star reviews** contribute **25%** (2,500 samples), followed closely by **5-star reviews** at **20%** (2,000 samples).
- **Lower ratings** (1-star and 2-star) make up a smaller portion of the dataset—**10% and 15%**, respectively.

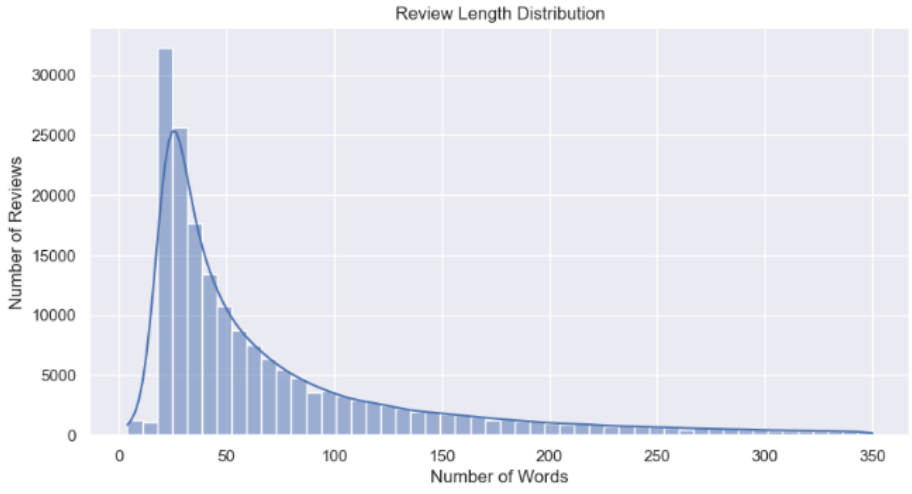
This imbalance reflects a **real-world scenario** where users are more likely to leave moderate-to-high ratings than low ratings

## 4.3 Review Length Distribution (Before Filtering)




**Insight:** Most reviews are under 100 words, but some extend beyond 5,000 words. The long tail justified filtering out extremely long reviews as outliers.

## 4.3 Review Length Distribution (After Filtering)



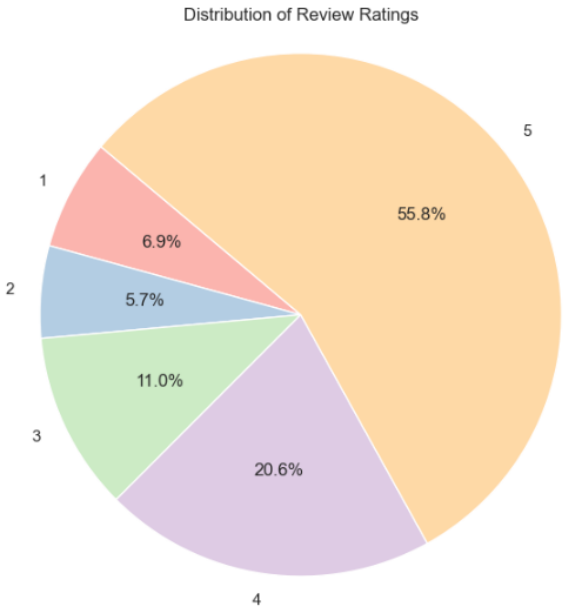
**Insight:** After filtering, the majority of reviews fall between 10 and 150 words, ensuring text inputs are neither too sparse nor too verbose.

## 4.5 Review Rating Distribution Using Piechart (Before Imbalancing)

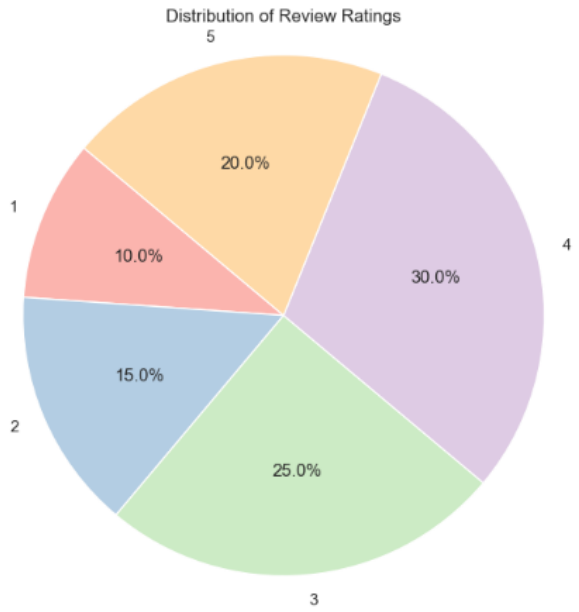
**Insight:**
The pie chart highlights a **highly imbalanced dataset**, with the majority of reviews skewed toward higher ratings:

- **5-star reviews** dominate at **55.8%** of the total.
- **4-star**: 20.6%
- **3-star**: 11.0%
- **2-star**: 5.7%
- **1-star**: 6.9%

This extreme imbalance means the model would be biased towards predicting higher ratings (especially 5 stars), resulting in poor performance when identifying negative or neutral reviews. Such imbalance undermines the model's generalizability and fairness.




## 4.2 Review Rating Distribution Using Piechart (After Imbalancing)




**Insight:**
This design introduces imbalance deliberately while preserving representation across all rating categories. It helps evaluate how models perform under unequal class conditions without being heavily dominated by any single class.

# 5. Train-Test Split

To evaluate the model's performance fairly and prevent overfitting, the cleaned and balanced dataset was split into training and testing subsets.

# Split training and testing data

```
X_train, X_test, y_train, y_test = train_test_split(df['reviewText'],df['Rating'], test_size=0.2, random_state=42,stratify=df['Rating'] )
```

### Split Strategy:

- **Method**: Stratified Train-Test Split
- **Tool**: `train_test_split()` from Scikit-learn
- **Split Ratio**:
  - **80% Training Set**
  - **20% Testing Set**
- **Stratification**: Ensured the proportion of each rating class (1 to 5 stars) remains equal in both training and test sets.

### Why Stratification?

Stratified sampling was used to **maintain class balance** in both the training and test sets. Without it, the test set might become imbalanced—even if the original dataset was balanced.

**Resulting Dataset Sizes:**

| Subset | | Number of Reviews |
|---|---|---|
| Training Set | - | 8,000 |
| Testing Set | - | 2,000 |

## 5.1Post-Split Preprocessing & Vectorization

After splitting the dataset into training and testing sets, additional natural language processing steps were applied to prepare the review text for machine learning.

### Stopword Removal & Lemmatization

To improve the semantic quality of the review text, two critical NLP operations were applied:

```
def spacy_preprocess(text)
   # Process the text using spaCy
   doc = nlp(text)
 # Filter and lemmatize
 tokens = [token.lemma_                  # spaCy lemmatization
      for token in doc
      if not token.is_stop and       # spaCy stopword removal
        not token.is_space and
        token.is_alpha            # Keep only alphabetic words (optional)
   ]
 return ' '.join(tokens)
#print the list of stopwords
print(sorted(nlp.Defaults.stop_words))
# Print total number of unique stopwords in spaCy's English model
print("Number of spaCy stopwords:", len(nlp.Defaults.stop_words))
```

```
["'d", "'ll", "'m", "'re", "'s", "'ve", 'a', 'about', 'above', 'across', 'after',
'afterwards', 'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'a
lso', 'although', 'always', 'am', 'among', 'amongst', 'amount', 'an', 'and', 'anot
her', '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', 'e
ight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'even', 'ever',
```

```
'every', 'everyone', 'everything', 'everywhere', 'except', 'few', 'fifteen', 'fift
y', 'first', 'five', 'for', 'former', 'formerly', 'forty', 'four', 'from', 'front'
, 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'have', 'he', 'hence', 'he
r', '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", 'n
ame', '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', 'oth
erwise', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'part', 'per', 'perhaps
', 'please', 'put', 'quite', 'rather', 're', 'really', 'regarding', 'same', 'say',
'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should'
, 'show', 'side', 'since', 'six', 'sixty', 'so', 'some', 'somehow', 'someone', 'so
mething', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'take', 'ten', 't
han', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'th
ereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'third
', 'this', 'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 't
o', 'together', 'too', 'top', 'toward', 'towards', 'twelve', 'twenty', 'two', 'und
er', 'unless', 'until', 'up', 'upon', 'us', 'used', 'using', 'various', 'very', 'v
ia', 'was', 'we', 'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever'
, 'where', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'wherever',
'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose'
, 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'your
s', 'yourself', 'yourselves', '‘d', '‘ll', '‘m', '‘re', '‘s', '‘ve', '’d', '’ll',
'’m', '’re', '’s', '’ve']

Number of spaCy stopwords: 326
```

*# Apply stopword removal and lemmatization to the training and test text data*

*# Apply preprocessing to each review in the training set*

*X_train = X_train.apply(spacy_preprocess)*

*# Apply preprocessing to each review in the test set*

*X_test = X_test.apply(spacy_preprocess)*

shows the stopword removal and lemmatized text:

```
['piece conjunction mount truck recently upgrade phone note large phone fit holder
not rest button'

 'base great review accuracy stylus ve decide try stylus thin comfortable hold sec
ond limit certain angel make light screechy sound write ipad screen tip stylus met
al like screechy sound blackboard fourth price stylus pretty steep consider issue
finally writing experience tiresome awkwardness stylus metal tip tap screen naviga
te ipad weird noisy yes easy write benefit marginal weigh con small benefit totall
y weight negative want like stylus point long screechy noise disappear d matter sc
reen sound will not away well styli fraction price stylus try different styli find
one mesh tip styli good accurate smooth smooth effortless write ipad look high rev
iew will not disappoint'

 'take trip new york girl grateful work need little charge small compact'

 'genuine samsung ac travel adapter identical come samsung note phone buy charger
home work catch weak battery need'
```

```
'look great not use flash camera cause come pink hassle']
```

## Stopword Removal:

- **Stopwords** are common words that occur frequently in language (e.g., “the”, “is”, “and”) but carry minimal semantic meaning.
- Removing them helps reduce noise and dimensionality in the vectorized feature space.
- In this project, stopwords were removed using:
    - **spaCy’s default English stopword set**, which offers linguistic richness and flexibility.

## Lemmatization:

- **Lemmatization** reduces each word to its base or **dictionary form** (called a "lemma").
- Unlike stemming (which can be more aggressive and less accurate), lemmatization ensures that words like:
    - `"running"`, `"ran"` → `"run"`
    - `"better"` → `"good"`
- This step helps group similar words and improves the model’s ability to generalize across different grammatical forms.
- Lemmatization was performed using **spaCy**, a fast and powerful NLP library in Python.

## About spaCy:

- SpaCy provides **tokenization**, **POS tagging**, **lemmatization**, **NER**, and **syntactic parsing**.
- The English language model used here is:

  ```
  python -m spacy download en_core_web_sm
  ```

  This model provides pre-trained weights for English NLP tasks, including lemmatization.

Note:

Both stopword removal and lemmatization were applied **after splitting** the data, and **separately on the training and testing sets**, to avoid **data leakage**. This ensures the model doesn't "peek" into the test set during training.

## 5.2 Text Vectorization (TF-IDF)

Once the text was cleaned and normalized, it needed to be converted into numerical form to be usable by machine learning models. For this, the **TF-IDF (Term Frequency–Inverse Document Frequency)** method was applied.

```
# Initialize and fit TF-IDF on training only
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
# Transform test using the same fitted vectorizer
X_test_tfidf = tfidf.transform(X_test)
```

### What is TF-IDF?

Term frequency-inverse document frequency (TF-IDF) is a metric you can use in machine learning to create a numerical representation of the words in a text document that demonstrates how relevant those words are within the text as a whole. TF-IDF is an important component of information retrievel and natural language processing because it can help AI models analyze the keywords and phrases within documents, providing a more nuanced understanding of what the text says.

Term frequency refers to how often a term is used in a body of text. Inverse document frequency looks at how many documents in a body of documents contain that term. To use a metaphor, term frequency considers how many times the word appears in the book, and inverse document frequency looks at how many books in the library use that term. In your local library, nearly every book would use words like “the” or “and” frequently. These words would have both a high term frequency (TF) and inverse document frequency (IDF), so an AI model would understand that those words weren’t particularly helpful for understanding what the document is about.

However, a word like “skydiving” would not appear in as many books. An AI model could use TF-IDF to determine that “skydiving” is a more important word for deciding what a document is about. Books with a higher TF for “skydiving” would be more relevant for people who want to check out a book about preparing for a skydive.

## What is TF-IDF used for?

You can use TF-IDF for three primary purposes: information retrieval, keyword extraction, and machine learning.

- **Information retrieval:** TF-IDF allows an AI model to retrieve information from a massive text library. For example, a search engine uses TF-IDF to determine which documents to provide as the result of a search query.
- **Keyword of feature extraction**: TF-IDF provides a mechanism for an AI model to determine the most important words in a text, which the model can then use to summarize the document.
- **Machine learning:** TF-IDF is an important part of natural language processing, a machine learning technique that allows computers and AI models to interpret human language. TF-IDF is important for training models to understand the patterns behind human language and the importance of individual words.

## Calculate term frequency and inverse document frequency

If you wanted to calculate TF-IDF, you would first need to calculate term frequency, then inverse document frequency, and finally TF-IDF. The equations for these calculations include three variables:

- t: term
- d: document
- D: set of documents

The equation to find term frequency is

= **tf(t,d)=log$f_0$(1+freq(t,d))**

The equation to find inverse document frequency
= **idf(t,D)=log$f_0$(N/count(d∈D:t∈d))**.

To put them together and find TF-IDF, the equation = **tfidf(t,d,D)=tf(t,d).idf(t,D).**

Your final results will measure how important the term is within the entire body of documents. The higher the score, the more relevant the AI model will consider that term.

**Vectorization Workflow in the Project:**

1. The `TfidfVectorizer` was **fitted only on the training data (**`X_train`**)** to learn the vocabulary and document frequencies.
2. The **same vectorizer** was then used to transform both:
    - `X_train` (fit and transform)
    - `X_test` (transform only)

Why Not Fit on Test Data?

- Fitting the vectorizer only on training data **prevents data leakage**.
- It simulates a real-world scenario where the model is trained on past data and evaluated on unseen data.

## Final Output Variables:

| Variable | Description |
|---|---|
| `X_train` | TF-IDF vectorized training data |
| `X_test` | TF-IDF vectorized testing data |
| `y_train` | Star ratings (1–5) for training reviews |
| `y_test` | Star ratings (1–5) for testing reviews |

This final processed dataset is now ready for **model training and evaluation**.

Machine Learning Algorithm