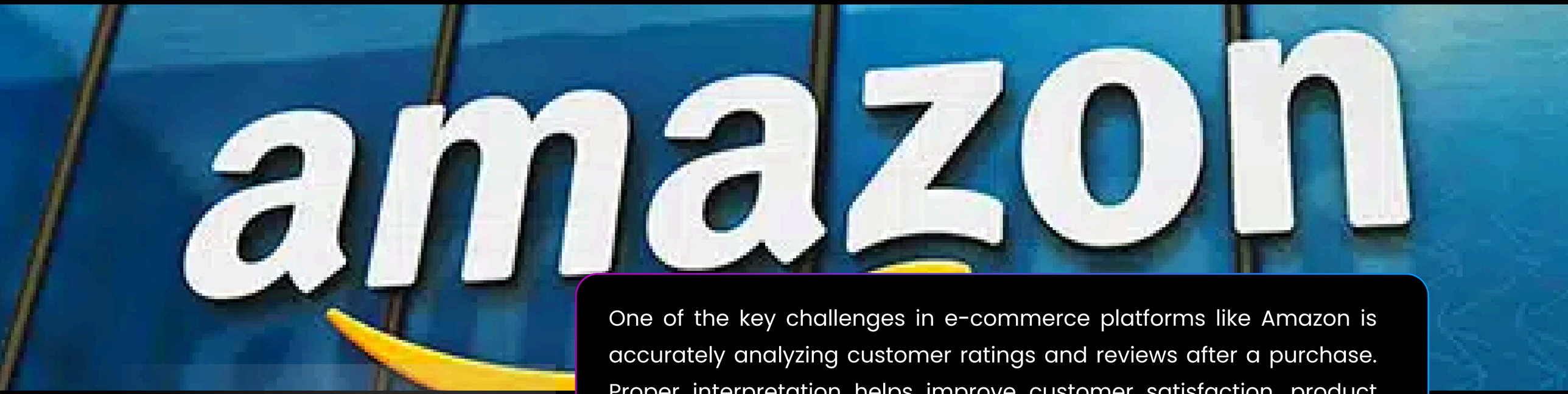
APPLICATION OF NLP FOR E-COMMERCE REVIEW SENTIMENT

By Raka Arrayan



Background

One of the key challenges in e-commerce platforms like Amazon is accurately analyzing customer ratings and reviews after a purchase. Proper interpretation helps improve customer satisfaction, product visibility, and overall shopping experience. Another concern is the ordering of reviews misleading comments can negatively impact both buyers and sellers. Addressing these issues through sentiment analysis can enhance trust, boost sales, and support better decision-making for all users.

Problem

- How to process user review data from Amazon so that it can be effectively used for sentiment analysis?
- What is the distribution of sentiments (positive, neutral and negative) in Amazon product reviews?
- Which method or technique is most suitable to classify sentiments from Amazon text reviews accurately?

Goals

- Perform data exploration to understand the characteristics and distribution of sentiments in Amazon reviews.
- Build a sentiment analysis model capable of classifying Amazon product reviews into positive, neutral ,and negative sentiments with good accuracy.
- Determine the optimal classification threshold to balance precision and recall for more reliable predictions

Dataset Overview

Dataset Characteritic

- Amount of data : 4,914 product review entries from Amazon
- Source: Kaggle
- Language: English
- Labeling: The dataset does not contain explicit sentiment labels, but they can be generated based on the overall rating score.
- Sentiment Proxy: The overall rating is used to classify sentiment as follows:
- Scores 4–5 → Positive
- Score 3 → Neutral
- Scores 1–2 → Negative

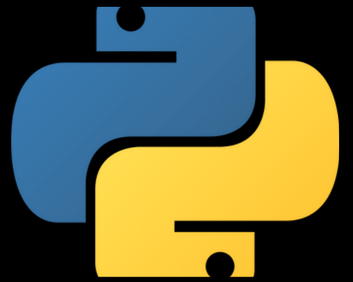
Key Features

- reviewerName : User Name
- **overall** : Product Rating
- **reviewText** : Evaluation Summary
- **reviewTime** : Evaluation Time
- **day_diff** : Number of days since assessment
- **helpful_yes** : The number of times the evaluation was found useful
- **helpful_no** : Number of people who didn't support the comment and didn't find it helpful
- **total_vote** : Number of votes given to the evaluation
- **score_pos_neg_diff** : score poz-neg

Tools

Google colab

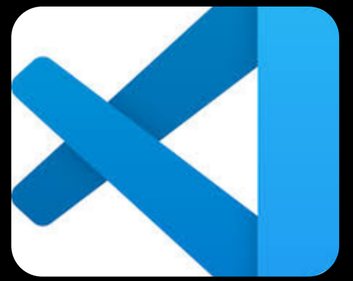
Microsoft Excel



Python

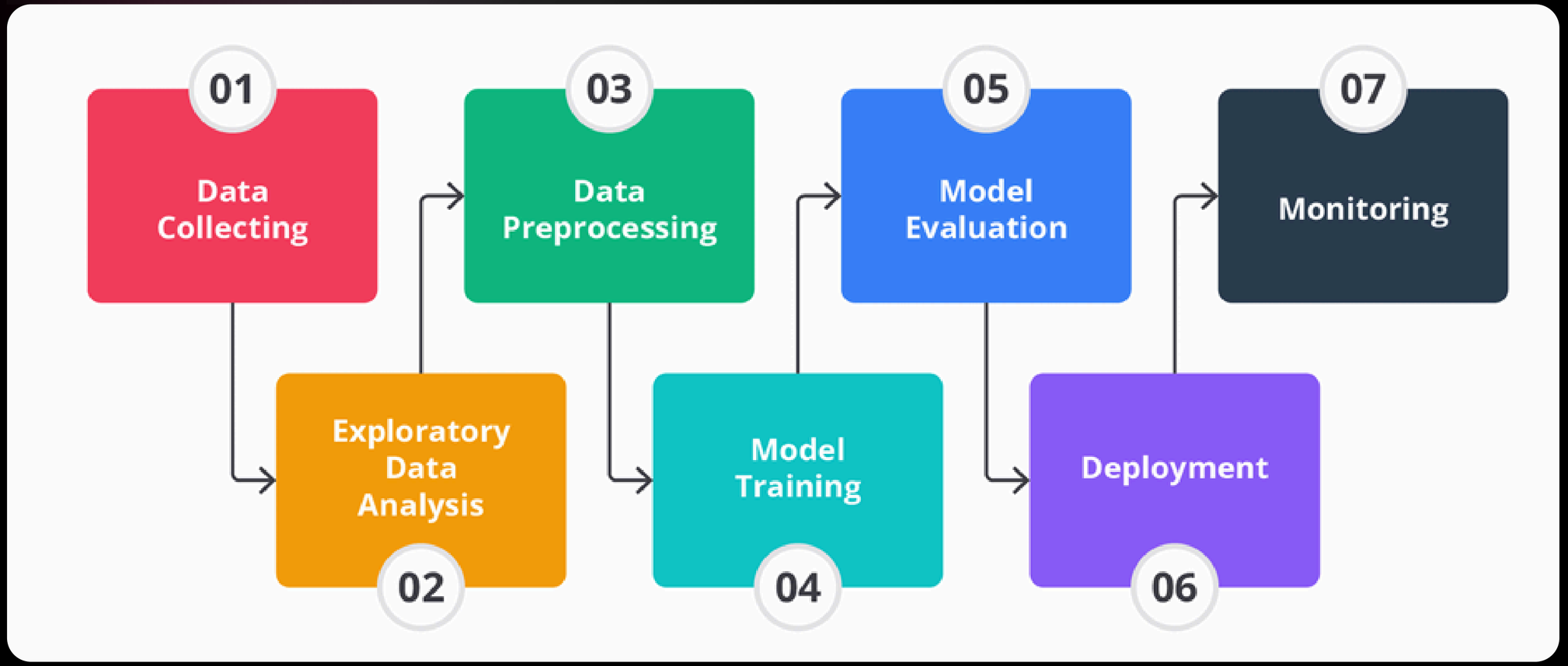
Github

Streamlit



Vscode

Data processing stages



DATA PREPARATION

Display of Data used

	reviewerN	overall	reviewText	reviewTime	day_diff	helpful_yes	helpful_no	total_vote	score_pos	score_ave	wilson_lower_bound
0		4	No issues.	7/23/2014	138	0	0	0	0	0	0
1	0mie	5	Purchased this for	10/25/2013	409	0	0	0	0	0	0
2	1K3	4	it works as expect	12/23/2012	715	0	0	0	0	0	0
3	1m2	5	This think has wor	11/21/2013	382	0	0	0	0	0	0
4	2&1/	5	Bought it with Ret	7/13/2013	513	0	0	0	0	0	0
5	2Cents!	5	It's mini storage.	4/29/2013	588	0	0	0	0	0	0

EXPLANATORY DATA ANALYSIS

Check missing value

	0
Unnamed: 0	0
reviewerName	1
overall	0
reviewText	1
reviewTime	0
day_diff	0
helpful_yes	0
helpful_no	0
total_vote	0
score_pos_neg_diff	0
score_average_rating	0
wilson_lower_bound	0

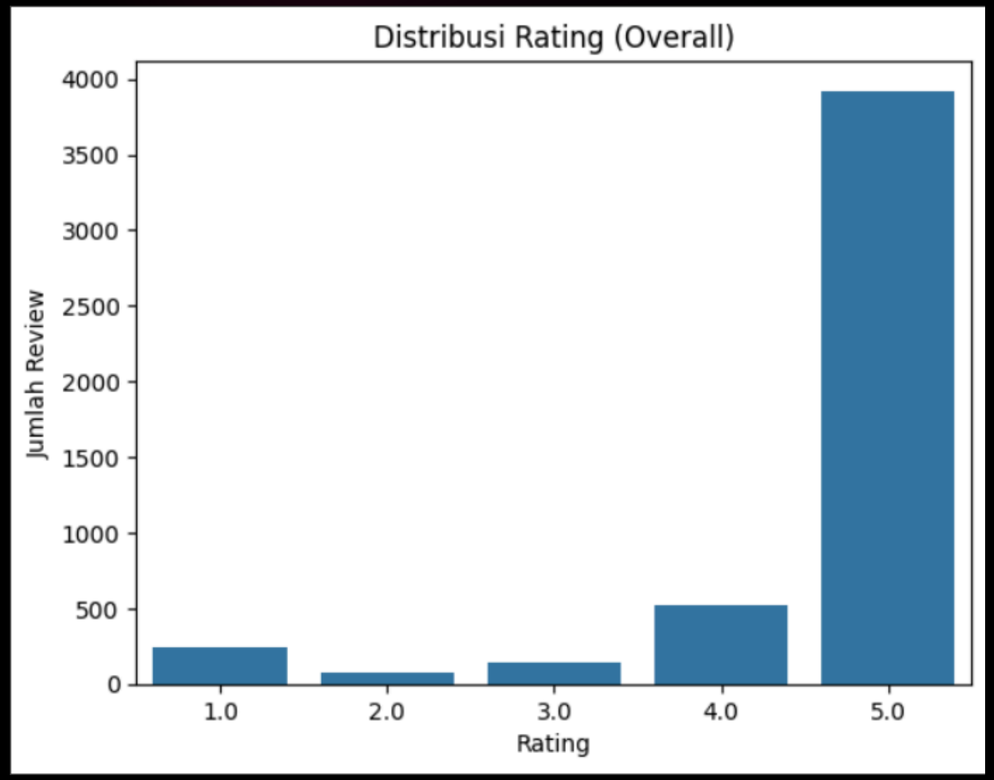
There are only two columns that have missing values, namely reviewerName and reviewText, each with 1 missing value.

Check missing value

	0
Unnamed: 0	0
reviewerName	0
overall	0
reviewText	0
reviewTime	0
day_diff	0
helpful_yes	0
helpful_no	0
total_vote	0
score_pos_neg_diff	0
score_average_rating	0
wilson_lower_bound	0

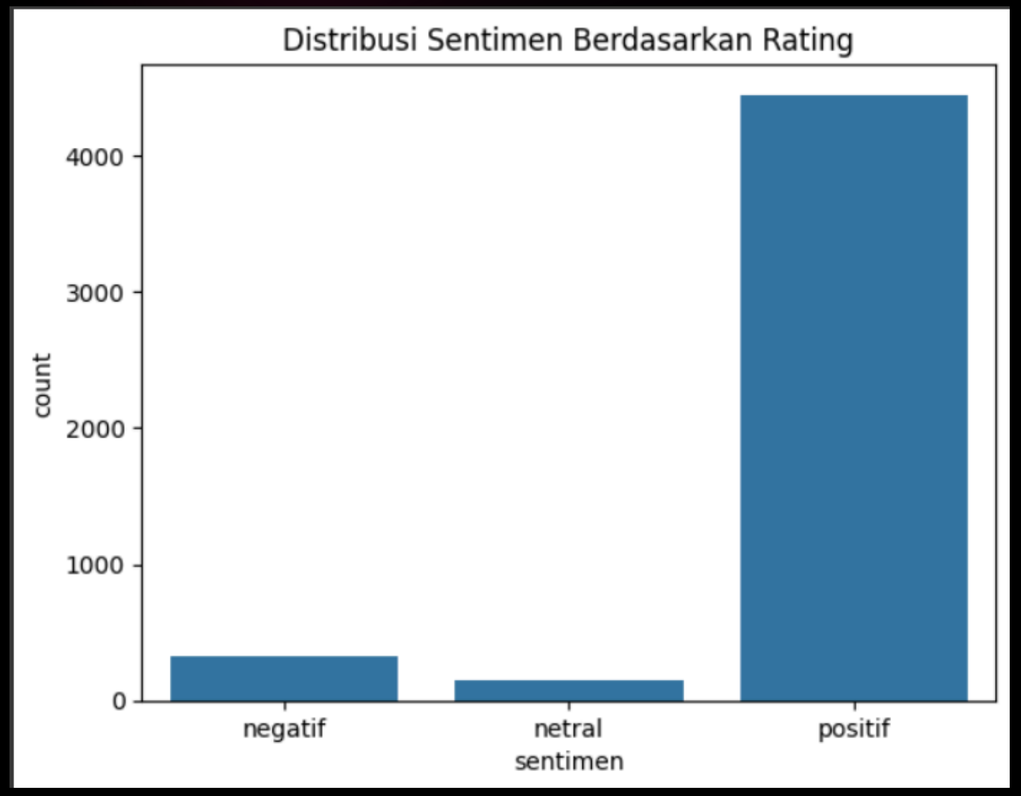
All rows that have missing values in reviewerName or reviewText are removed, so that a recheck with df.isna().sum() shows that there are no more missing values in all columns.

Check Rating Distribution



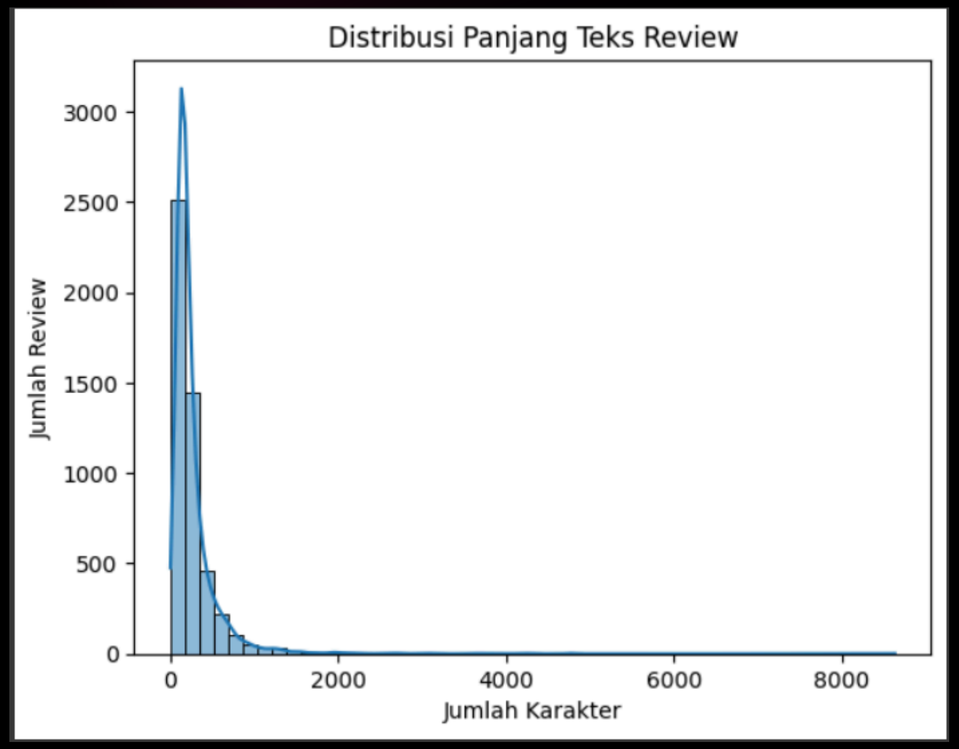
The graph below shows that the majority of users gave a rating of 5, with nearly 4,000 reviews, significantly higher than other ratings. Meanwhile, a rating of 4 came in second, followed by ratings of 1, 3, and 2, which were much smaller in number. This distribution pattern indicates a tendency for users to give very positive ratings, which could indicate high satisfaction or bias in the reviews.

Create a Sentiment Column (Labeling) from the overall and see its distribution



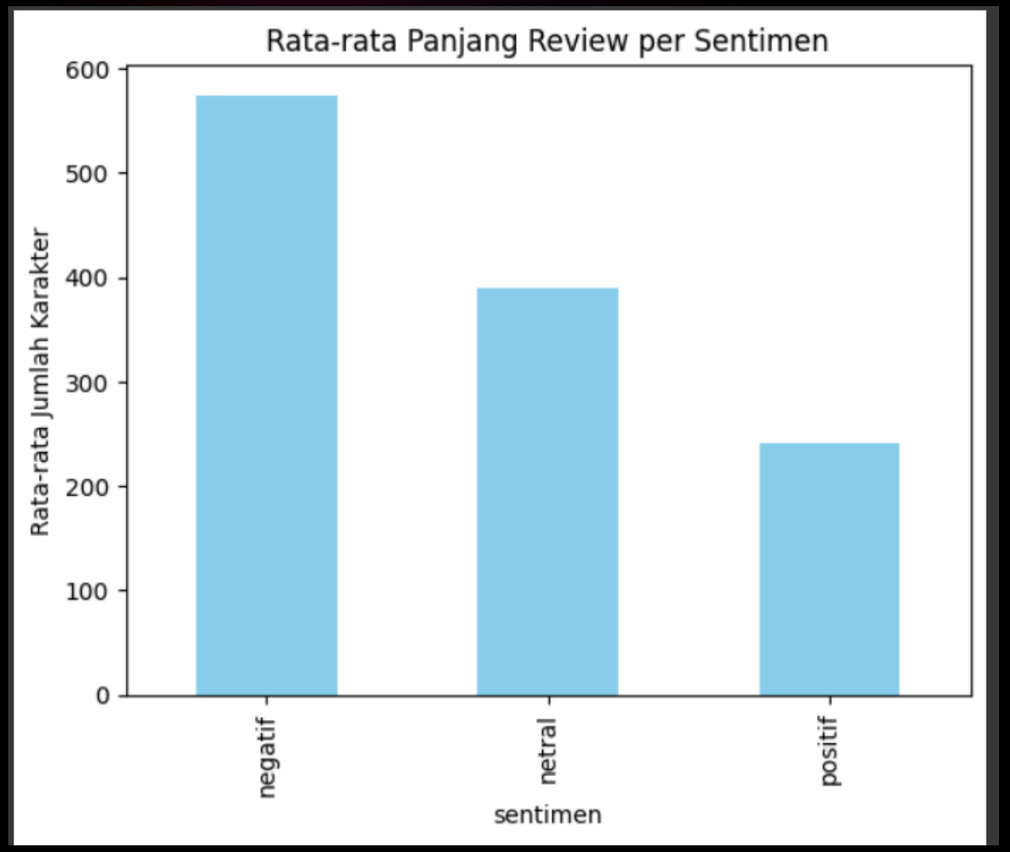
The figure below shows the distribution of sentiment based on ratings classified using the label_sentiment() function, where ratings ≤ 2 are considered negative, ratings 3 as neutral, and ratings ≥ 4 as positive. The visualization using sns.countplot() shows that the majority of reviews are in the positive sentiment category, with the number far exceeding the negative and neutral categories. This indicates that most users provide satisfactory reviews.

Add a Review Text Length Column and see its distribution



The figure below shows the distribution of review text lengths based on the number of characters calculated using len() in the reviewText column. The visualization results using a histogram (sns.histplot) and a KDE (Kernel Density Estimation) curve show that most reviews are relatively short, typically under 500 characters, with a peak in the 100–200 character range. This distribution is right-skewed.

Viewing Average Review Length per Sentiment Label



The figure below displays the average length of review texts by sentiment category, calculated by grouping the data using groupby('sentiment') and taking the average value from the review_length column. The visualization shows that reviews with negative sentiment have the highest average character count.This indicates that users tend to write longer reviews when leaving negative reviews, likely because they want to explain their complaints in detail, while positive reviews tend to be short and to the point with praise.

DATA PREPCESSING

Cleaning Text

```python
import re
import nltk
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

# cleaning text
def clean_text(text):
    # 1. Ubah ke huruf kecil
    text = text.lower()
    # 2. Hapus tanda baca dan angka
    text = re.sub(r'[^a-z\s]', '', text)
    # 3. Hapus stopwords
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text
```

The clean_text() function cleans text by lowercase letters, removing punctuation and numbers, and removing stopwords to make the text neater and ready for analysis

```python
def rewhitespace(text):
    corrected = str(text)
    corrected = re.sub(r"( )\1+", r"\1", corrected)
    corrected = re.sub(r"(\n)\1+", r"\1", corrected)
    corrected = re.sub(r"(\r)\1+", r"\1", corrected)
    corrected = re.sub(r"(\t)\1+", r"\1", corrected)
    return corrected.strip()
```

The rewhitespace() function cleans text from repeated spaces, newlines (\n), carriage returns (\r), and tabs (\t), then removes spaces at the beginning and end of the text to make it neater.

Tokenization

```
# Hapus resource punkt yang rusak
import shutil
shutil.rmtree('/root/nltk_data/tokenizers/punkt', ignore_errors=True)

# Unduh ulang resource punkt
import nltk
nltk.download('punkt')
```

Delete the corrupted NLTK resource "punkt" and then re-download it so it can be used again for normal text tokenization.

```
# Import dan lakukan tokenisasi
from nltk.tokenize import word_tokenize

def token(text):
    return word_tokenize(text)

df_final['tokenized_review'] = df_final['clean_review'].apply(lambda x: x.split())
df_final.head()
```

word_tokenize() is more accurate because it processes punctuation, but it can't be used due to errors. Therefore, x.split() is used as a simple alternative to splitting words.

Tokenization

	sentimen	clean_review	tokenized_review
1	positif	purchased device worked advertised never much ...	[purchased, device, worked, advertised, never,...
2	positif	works expected sprung higher capacity think ma...	[works, expected, sprung, higher, capacity, th...
3	positif	think worked greathad diff bran gb card went s...	[think, worked, greathad, diff, bran, gb, card...
4	positif	bought retail packaging arrived legit orange e...	[bought, retail, packaging, arrived, legit, or...
5	positif	mini storage doesnt anything else supposed pur...	[mini, storage, doesnt, anything, else, suppos...

The tokenization results show that each text in the clean_review column has been successfully broken down into a list of words (tokens) in the tokenized_review column. For example, the sentence "purchased device worked advertised never..." is converted into a list like [purchased, device, worked, advertised, never, ...], which facilitates further text analysis.

Lemmatization

	sentimen	clean_review	tokenized_review	lemmatized_review
1	positif	purchased device worked advertised never much ...	[purchased, device, worked, advertised, never,...	[purchase, device, work, advertised, never, mu...
2	positif	works expected sprung higher capacity think ma...	[works, expected, sprung, higher, capacity, th...	[work, expect, sprung, high, capacity, think, ...
3	positif	think worked greathad diff bran gb card went s...	[think, worked, greathad, diff, bran, gb, card...	[think, work, greathad, diff, bran, gb, card, ...
4	positif	bought retail packaging arrived legit orange e...	[bought, retail, packaging, arrived, legit, or...	[bought, retail, packaging, arrive, legit, ora...
5	positif	mini storage doesnt anything else supposed pur...	[mini, storage, doesnt, anything, else, suppos...	[mini, storage, doesnt, anything, else, suppos...

The lemmatization results show that each word in the tokenized_review has been converted to its base form in the lemmatized_review column. For example, the word "purchased" becomes "purchase" and "worked" becomes "work." This process helps simplify word variations, making text analysis more consistent and accurate.

Preparing Text Data from Lemmatization Results

	sentimen	lemmatized_review
1	positif	[purchase, device, work, advertised, never, mu...
2	positif	[work, expect, sprung, high, capacity, think, ...
3	positif	[think, work, greathad, diff, bran, gb, card, ...
4	positif	[bought, retail, packaging, arrive, legit, ora...
5	positif	[mini, storage, doesnt, anything, else, suppos...

In this section, I remove the clean_review and tokenized_review columns from the df_final DataFrame using the drop() function, after that, I display the DataFrame using df_final.head() to ensure that only the sentiment and lemmatized_review columns remain, and are ready to be used for further analysis.

Training models

Tf-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer #mengubah teks menjadi angka berdasarkan tf-idf

X = df_final['lemmatized_review']# fitur input
y = df_final['sentimen']# target atau label

tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2), stop_words='english')
# Hanya mengambil 5000 fitur lalu gunakan unigram(kata tunggal) dan bigram(pasangan kata) selanjutnya hapus stopwords
review_tf = tfidf_vectorizer.fit_transform(X.astype('U'))
#memastikasn semua data dalam tipe string unicode dan akan kita train dari teks untuk membuat tf-idf
review_tf
```

Transforming the text in the 'lemmatized_review' column into a numeric representation using TF-IDF by considering unigrams and bigrams, limiting the number of features to 5000, and removing English stopwords, so that the text is ready to be used as input for a machine learning model.

Handle imbalance data

```python
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(review_tf, y)
```

```
sentimen
positif    4447
netral     4447
negatif    4447
Name: count, dtype: int64
```

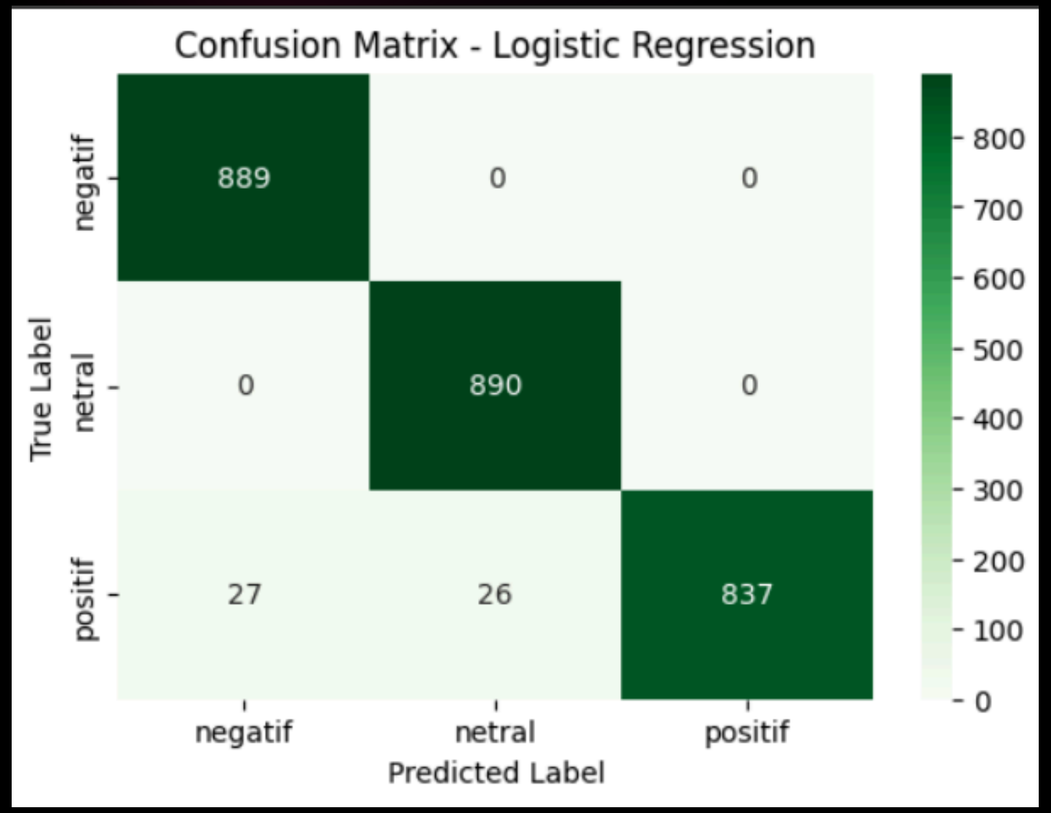
Perform oversampling using RandomOverSampler from the imblearn library to address data imbalance in sentiment labels. With this method, the number of samples for each class (positive, neutral, and negative) is balanced to 4447 data each, so that the machine learning model is not biased towards the majority class when trained.

Training and Evaluation of Logistic Regression with Oversampling Data

```
Logistic Regression Accuracy:  0.9801423754215062
Logistic Regression Precision:  0.9807132459970888
Logistic Regression Recall:  0.9801498127340823
Logistic Regression F1 Score:  0.9799853462742844

Confusion Matrix:
[[889   0   0]
 [  0 890   0]
 [ 27  26 837]]
--------------------------------------------------------
              precision    recall  f1-score   support

     negatif       0.97      1.00      0.99       889
      netral       0.97      1.00      0.99       890
     positif       1.00      0.94      0.97       890

    accuracy                           0.98      2669
   macro avg       0.98      0.98      0.98      2669
weighted avg       0.98      0.98      0.98      2669
```

The Logistic Regression model evaluation results demonstrated excellent sentiment classification performance. Overall accuracy reached 98%, with high precision, recall, and F1-score across all classes. The confusion matrix shows that the model performed nearly perfectly in classifying negative and neutral sentiment, but still made some errors in classifying positive sentiment (e.g., 27 and 26 positive data points were incorrectly classified as negative and neutral, respectively).

Training and Evaluation of Logistic Regression with Oversampling Data



This matrix shows that the model performed very well on negative and neutral sentiment data, with 889 and 890 correct predictions, respectively, with no misclassifications. However, on positive sentiment, there were several errors: 27 positive data points were classified as negative, and 26 data points were classified as neutral, while 837 were correctly classified as positive.

Save model

Saves the machine learning model and TF-IDF vectorizer to a file

```
import pickle

# Simpan TF-IDF Vectorizer
with open('tfidf_vectorizer.sav', 'wb') as f:
    pickle.dump(tfidf_vectorizer, f)
print("TF-IDF Vectorizer saved to 'tfidf_vectorizer.sav'")

# Simpan Logistic Regression Model
with open('logistic_regression_model.sav', 'wb') as f:
    pickle.dump(clf_lr, f)
print("Logistic Regression model saved to 'logistic_regression_model.sav'")
```

The code saves the TF-IDF Vectorizer and Logistic Regression model objects to files using the pickle module, allowing the trained models and vectorizations to be reused without retraining. The tfidf_vectorizer.sav file stores the text transformation, while the logistic_regression_model.sav file stores the classification model.

load machine learning model and TF-IDF vectorizer from .sav file

```python
import pickle

# Load TF-IDF Vectorizer
with open('tfidf_vectorizer.sav', 'rb') as f:
    tfidf_vectorizer = pickle.load(f)
print("TF-IDF Vectorizer loaded successfully.")

# Load Logistic Regression Model
with open('logistic_regression_model.sav', 'rb') as f:
    clf_lr = pickle.load(f)
print("Logistic Regression model loaded successfully.")
```

This code is used to reload the TF-IDF Vectorizer and Logistic Regression model objects previously saved using pickle. By opening the .sav file in 'rb' (read binary) mode, pickle.load() returns the objects to Python memory. Once loaded, these two objects are ready to be used for new text transformations and predictions without the need for retraining.

Testing the saved and loaded model

```
Test Accuracy:  0.9741502137187055
Test Precision:  0.8491877944028817
Test Recall:  0.9904804737276066
Test F1 Score:  0.9098687878115067

Confusion Matrix:
[[ 324    0    0]
 [   0  142    0]
 [  77   50 4320]]

Classification Report:
              precision    recall  f1-score   support

     negatif       0.81      1.00      0.89       324
      netral       0.74      1.00      0.85       142
     positif       1.00      0.97      0.99      4447

    accuracy                           0.97      4913
   macro avg       0.85      0.99      0.91      4913
weighted avg       0.98      0.97      0.98      4913
```

The saved and reloaded model still performed well, with an accuracy of 97.4%. Recall was very high (99%), but precision was lower (84.9%) due to some misclassifications, particularly in the positive class. Overall, the model remained strong, with an F1-score of 90.9%.

Raka Arrayan

Home

About

Contact

amazon

Deployment on Streamlit

E-Commerce Review Sentiment Analyzer

Unlock insights from product reviews using Natural Language Processing (NLP)

Enter an E-Commerce Product Review

Please enter your product review below. The analysis works best with English reviews.

The product shipped fast and actually came in a few days early. I read about some people having problems with this card. When I got it, first thing I did was format it. Also make sure your device is compatible with SDHC cards. I have had no problems. Very fast read/write speeds. A little over 29 GB actually available for storage. I am pleased.

Classify

The sentiment is POSITIVE.

E-Commerce Review Sentiment Analyzer

Unlock insights from product reviews using Natural Language Processing (NLP)

Enter an E-Commerce Product Review

Please enter your product review below. The analysis works best with English reviews.

Lasted about 8 months of daily use in my Android phone. But yesterday it stopped performing writes and then failed altogether. While it was working, reads were incredibly fast and writes were pretty good as well.

Classify

The sentiment is NEUTRAL.

E-Commerce Review Sentiment Analyzer

Unlock insights from product reviews using Natural Language Processing (NLP)

Enter an E-Commerce Product Review

Please enter your product review below. The analysis works best with English reviews.

Yeah. Ultra Hot. I could not get any of my devices to recognize this card. When I finally was, using a USD reader, I tried to format it. It wouldn't format and burned my fingers when I removed it. Others have also posted this. Maybe they are not all bad, but many are. Perhaps they are counterfeit, I don't know. But the hassle of having to return it isn't worth the $10 you may save.

Classify

The sentiment is NEGATIVE.

Thank You

FOR YOUR ATTENTION

Link Github