2) Customer Feedback & Sentiment Dataset

TC. Antony

Project Observation:

Project Observations: Customer Feedback & Sentiment Dataset

Context:
The Customer Feedback & Sentiment Dataset is used for sentiment analysis of customer complaints and feedback. The goal is to derive insights that can help improve service quality by understanding customer satisfaction and identifying pain points.

Dataset Overview:
The dataset consists of the following variables:

- **Review_ID:** A unique identifier for each customer review.
- **Customer_ID:** A unique identifier for the customer providing the feedback.
- **Review_Text:** The textual content of the customer's feedback or complaint.
- **Sentiment_Label:** A categorical variable representing the sentiment of the review — classified into Positive, Negative, or Neutral.
- **Rating:** A numerical rating provided by the customer, likely reflecting their overall satisfaction.
- **Service_Type:** The category of service the feedback pertains to, such as Claim, Policy Purchase, or Support.

Observations:

1. **Sentiment Analysis Scope:** The dataset enables sentiment classification by leveraging both structured data (like Rating and Service_Type) and unstructured data (Review_Text). This allows for a comprehensive understanding of customer sentiment.
2. **Textual Data Importance:** The Review_Text column serves as a critical input for Natural Language Processing (NLP) techniques. It captures the qualitative aspect of customer feedback, which often provides deeper context than numerical ratings.
3. **Categorical Sentiment Labels:** The sentiment labels — Positive, Negative, and Neutral — provide clear target classes for supervised learning models.
4. **Service Segmentation:** The Service_Type feature helps segment feedback based on different service interactions, allowing service-specific sentiment analysis and identifying service areas needing improvement.
5. **Potential Model Inputs:** To predict sentiment, the Review_Text column will likely be transformed into numerical embeddings (like BERT embeddings), while Rating and Service_Type could be used as additional features. Alternatively, the model can focus solely on textual input for sentiment prediction.
6. **Enhancing Model Performance:** Given the presence of categorical and numerical features, exploring feature engineering techniques, such as encoding Service_Type and scaling Rating, could further enhance model performance when combining them with text-based embeddings.

These observations provide a solid foundation for developing and refining sentiment analysis models aimed at improving customer satisfaction and service quality.

1 - EDA - [7 - 20 Slide]
2 - SQL - [21 - 35 Slide]

3 - Preprocessing - [36 - 61 Slide]

I] Missing Values
II] Sentiment Labeling
III] Text Classification Tokenization

i] Taken Review_Text Column for Text Classification
ii] Stopwords Removal
iii] Punctuation and Special Character Removal
iv] Tokenization
v] Lemmatization

6] Future Prediction - 2 [76 - 86 Slide]

7] Streamlit Application [88 - 92 Slide]

Previous using Feature has 3 Columns that has Wrong so now Am Using to Predict Review_Text for Only for Prediction Using BERT Embedding

8] Feature Selection / Model Training - 2
[93 - 125 Sheat]

9] Preprocessing / Feature Selection - 3
[126 - 140 Sheet]

10] Model Training - 3 [141 - 146 Sheet]
11] Future Prediction -3 [147 - 150 Sheet]
12] My Experience [151 Sheet]
13] Conclusion [152 Sheet]

Dataset:

Review_ID	Customer_ID	Review_Text	Sentiment_Label	Rating	Service_Type
bdd640fb-0667-4ad1-9c80-317fa3b1799d	23b8c1e9-3924-46de-beb1-3b9046685257	Beautiful instead ahead despite measure ago cu...	Neutral	1	Claim
12476f57-a5e5-45ab-aefc-fad8efc89849	88bd6407-2bcf-4e01-a28d-efe39bf00273	Left establish understand read. Range successf...	Neutral	3	Claim
cac5b68c-28f4-4481-a0a0-4dc427209bdf	10435a10-98ae-4334-ac12-ace8ae340454	Other life edge network wall quite. Race Mr en...	Positive	2	Customer Support
913e4de2-e0c5-4cb8-bda9-c2a90ed42f1a	bb5e4bcf-15ed-4269-9429-6c07f26b4776	Within mouth call process. Close month parent ...	Positive	5	Claim
dfde4fbf-3ff3-40bf-b66e-cb15474ebc19	ceda8bbb-7171-4434-934c-6c92ec5b227c	Anything yourself structure why. Coach magazin...	Neutral	4	Claim

```
[ ]  1 df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Unnamed: 0       1000 non-null   int64
 1   Review_ID        1000 non-null   object
 2   Customer_ID      1000 non-null   object
 3   Review_Text      1000 non-null   object
 4   Sentiment_Label  1000 non-null   object
 5   Rating           1000 non-null   int64
 6   Service_Type     1000 non-null   object
dtypes: int64(2), object(5)
memory usage: 54.8+ KB
```

```
1 df2.nunique()
```

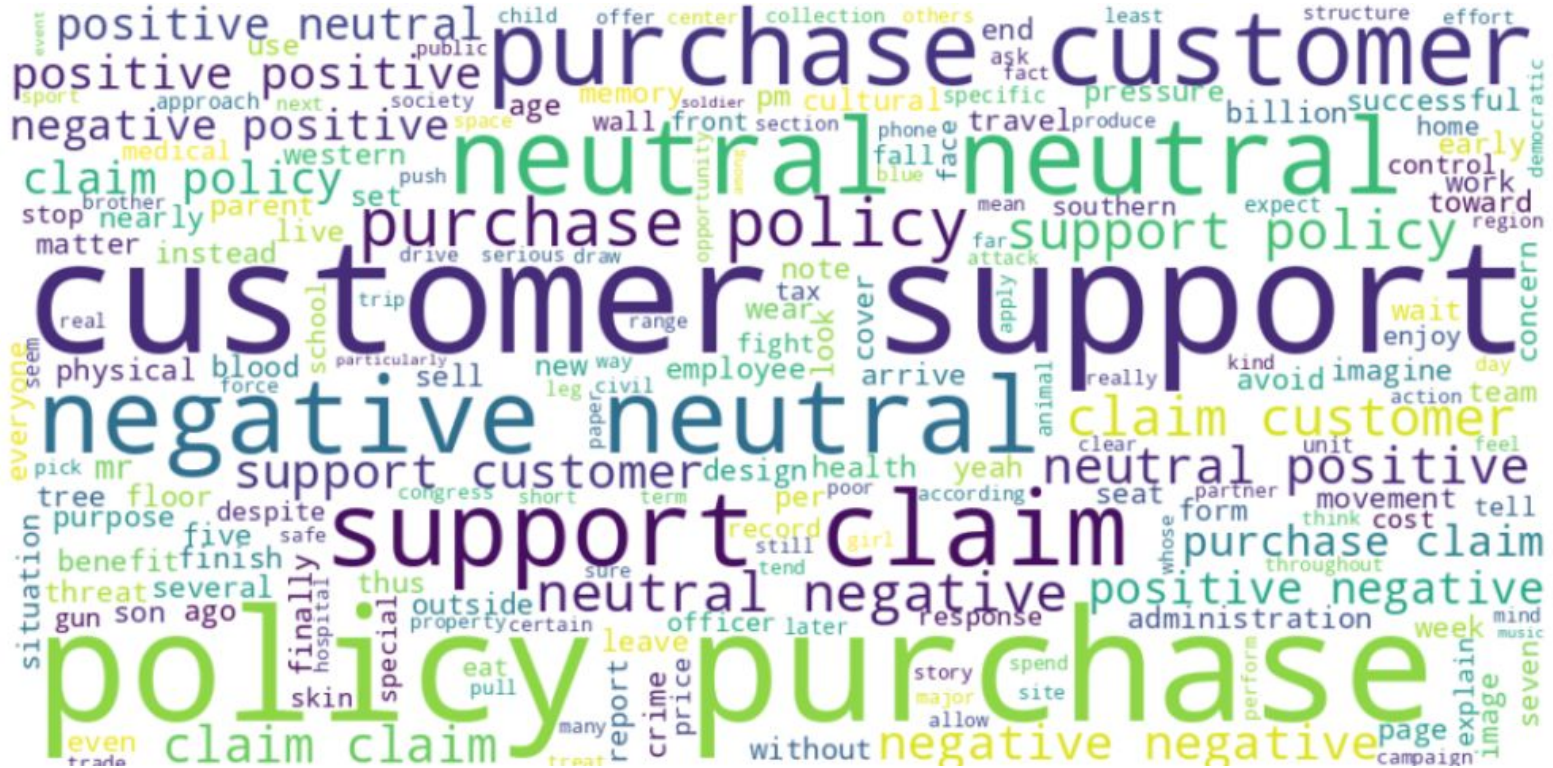
	0
Review_ID	1000
Customer_ID	1000
Review_Text	1000
Sentiment_Label	3
Rating	5
Service_Type	3

dtype: int64

```
[ ]  1 # Identify categorical and numerical columns
     2 categorical_columns = df2.select_dtypes(include=['object']).columns.tolist()
     3 numerical_columns = df2.select_dtypes(include=['int64', 'float64']).columns.tolist()
     4
     5 print("Categorical Columns:", categorical_columns)
     6 print("Numerical Columns:", numerical_columns)
```

```
Categorical Columns: ['Review_ID', 'Customer_ID', 'Review_Text', 'Sentiment_Label', 'Service_Type']
Numerical Columns: ['Rating']
```

Create word cloud for categorical column for better Understanding



Description: Contains customer feedback, complaints, and sentiments extracted from reviews and social media.

Best Cross-Check Charts:

Sentiment vs. Rating – Boxplot to analyze how ratings vary across sentiments.

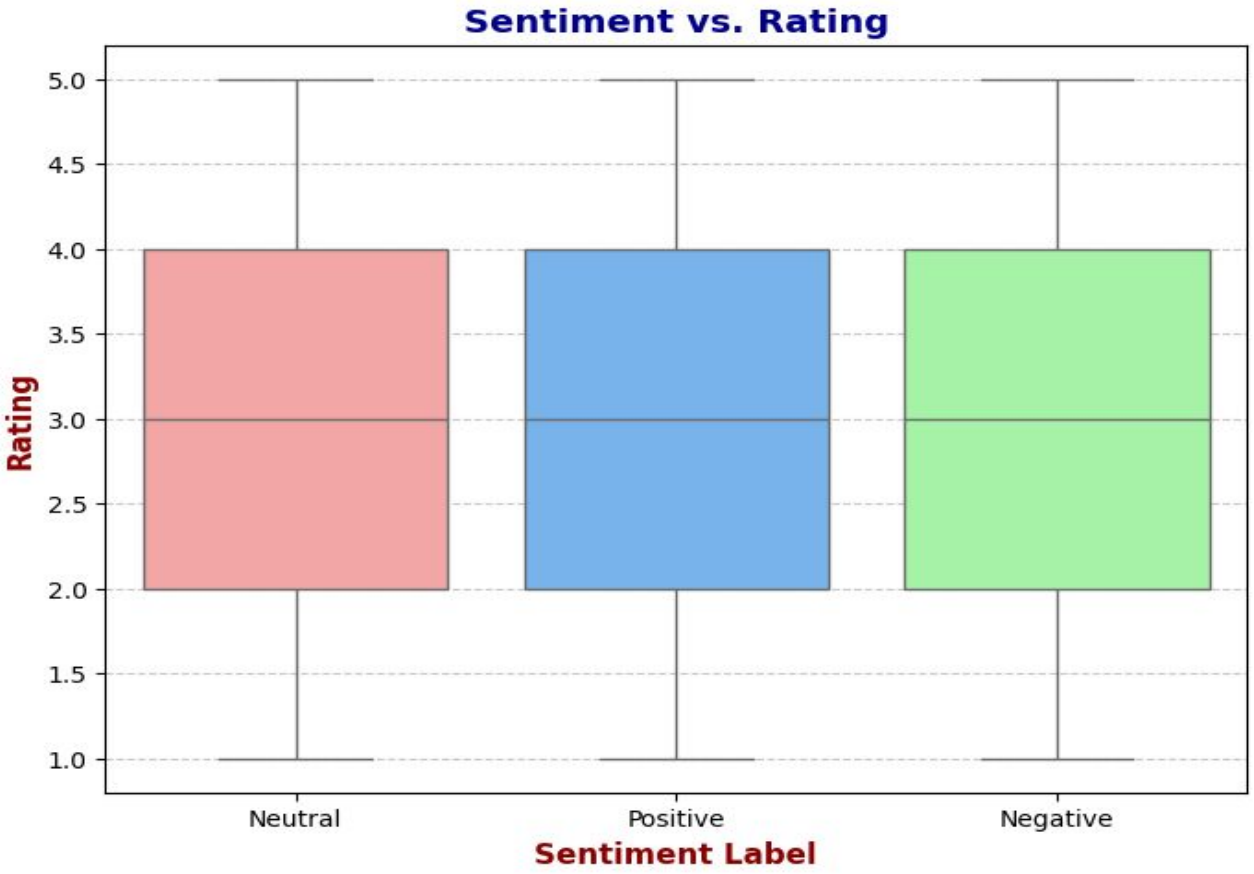
Service Type vs. Rating – Boxplot to check service performance.

Sentiment Distribution – Count plot to visualize sentiment frequency.

Rating Distribution – Histogram to understand rating spread.

Customer Reviews Frequency – Bar plot to identify repeat reviewers.

1) Sentiment_Label vs. Rating



Sentiment_Label	count
Neutral	354
Negative	331
Positive	315

dtype: int64

Rating	count
4	220
3	205
1	198
2	189
5	188

Insights from the Box Plot (Sentiment vs. Rating):

Weak Positive Correlation: While there's a trend suggesting that positive sentiment tends to be associated with slightly higher ratings, and negative sentiment with slightly lower ratings, the relationship isn't very strong.

Significant Overlap: The rating distributions for all three sentiment categories (positive, negative, and neutral) overlap considerably. This means you can't reliably predict a rating based on sentiment alone.

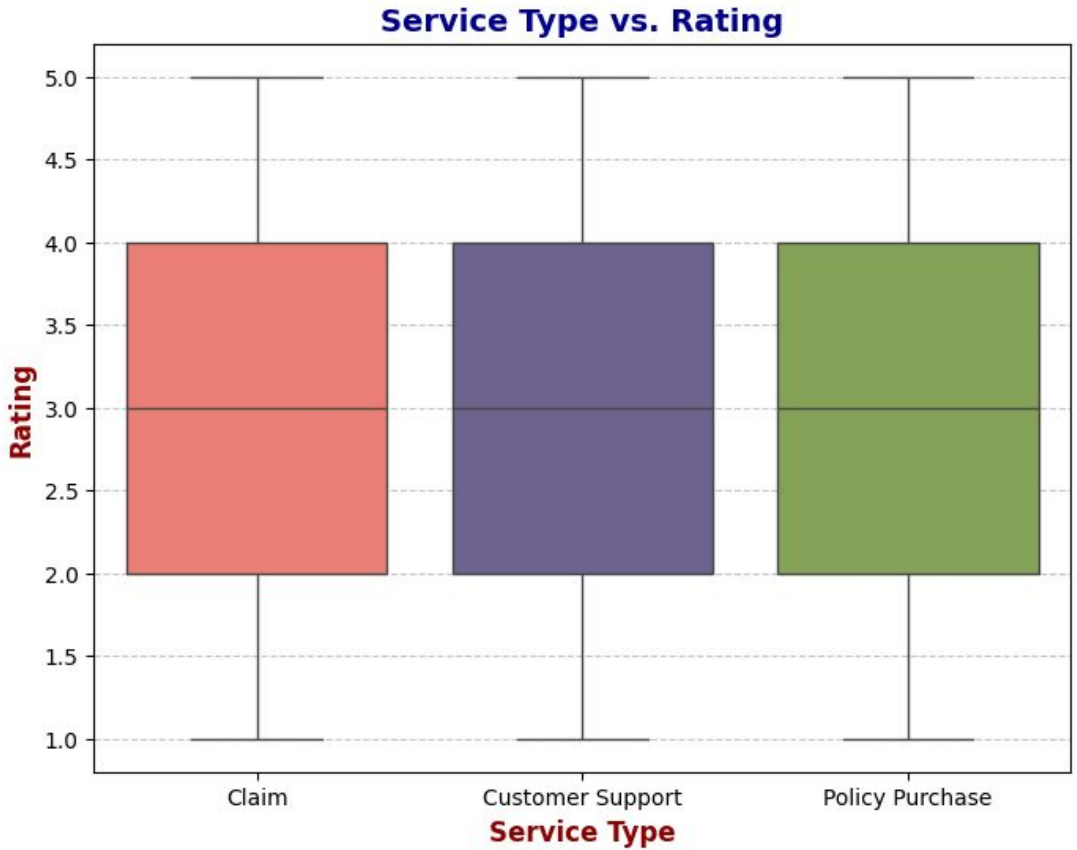
Sentiment is Not Deterministic:

A positive sentiment doesn't guarantee a high rating,

and a **negative sentiment** doesn't always lead to a low rating.

Neutral Sentiment's Variability: The neutral category shows a wide range of ratings, suggesting that neutral sentiment doesn't necessarily indicate a consistent level of satisfaction.

2) Service Type vs Rating



Service_Type	count
Customer Support	339
Policy Purchase	335
Claim	326

Key Observations and Insights:

Overall Trend:

Similar to the previous sentiment analysis, there's a trend, but it's not dramatically strong. The ratings for all three service types (Claim, Customer Support, and Policy Purchase) are relatively close.

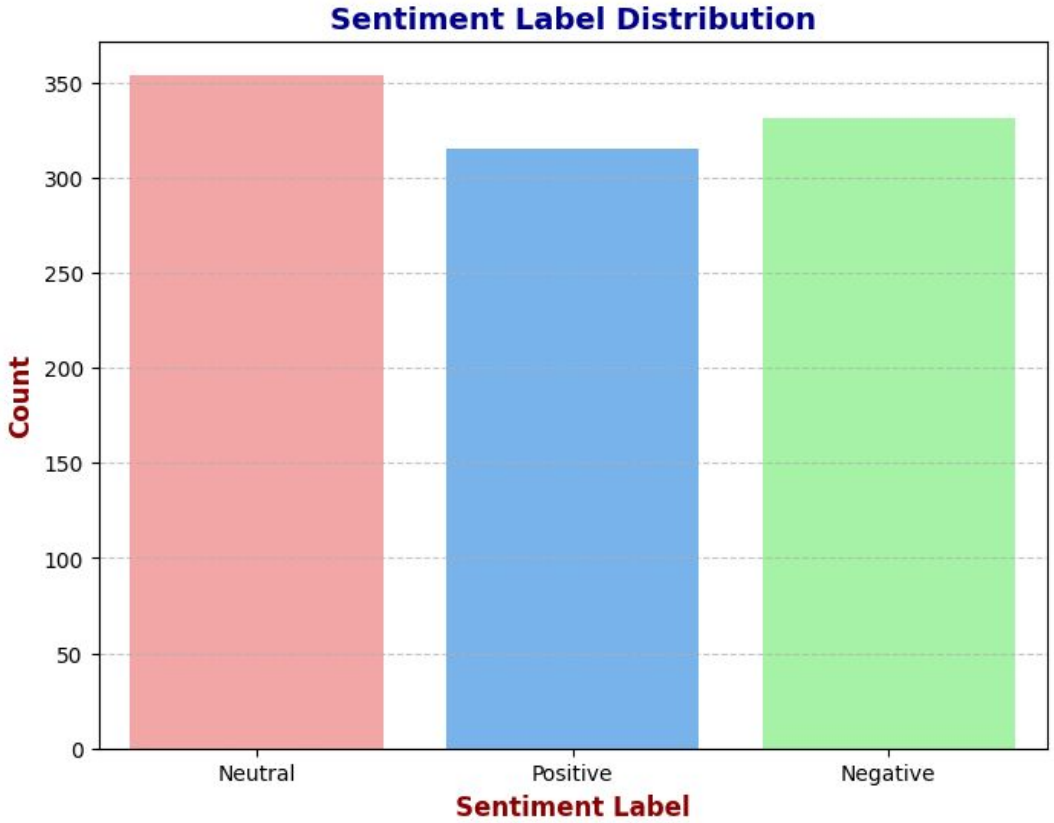
Claim Service Slightly Lower:

The "Claim" service type seems to have a slightly lower median rating compared to the other two. This suggests that customers may be less satisfied with the claim process than with customer support or policy purchase.

Customer Support and Policy Purchase Similar:

The **distributions for "Customer Support" and "Policy Purchase" are very similar**, with overlapping boxes and medians. This indicates that customers generally rate these two services comparably.

3) Sentiment Distribution



Sentiment_Label	count
Neutral	354
Negative	331
Positive	315

dtype: int64

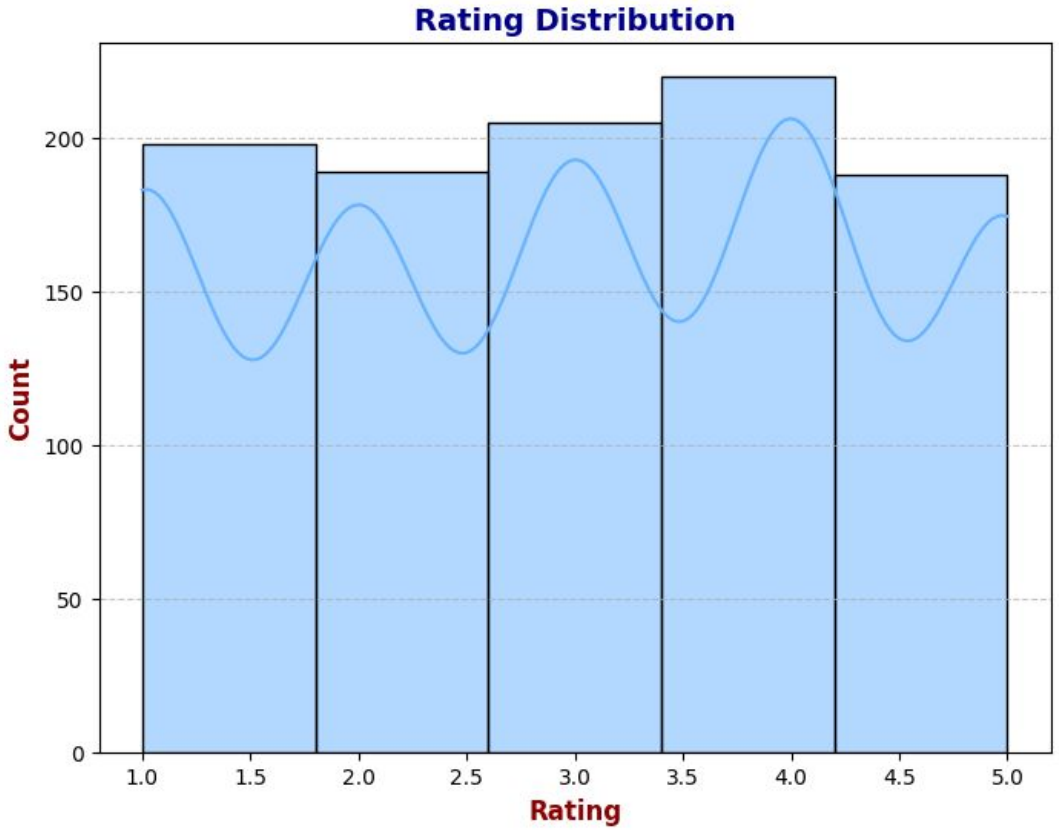
Neutral Dominance: The **"Neutral" sentiment category has the highest count,** indicating a large portion of the data is classified as neutral.

Comparable Positive and Negative: The counts for "Positive" and "Negative" sentiments are relatively close, suggesting a similar level of positive and negative feedback.

Imbalanced Distribution: The distribution is somewhat imbalanced, with "Neutral" significantly higher than the other two.

Potential for Further Analysis: The high neutral count might warrant investigation into the reasons for neutrality (e.g., ambiguous language, mixed feelings).

4) Rating Distribution



Rating	count
4	220
3	205
1	198
2	189
5	188

Relatively Uniform Distribution:

The ratings are distributed relatively evenly across the **scale (1 to 5)**, with **no single rating dominating**.

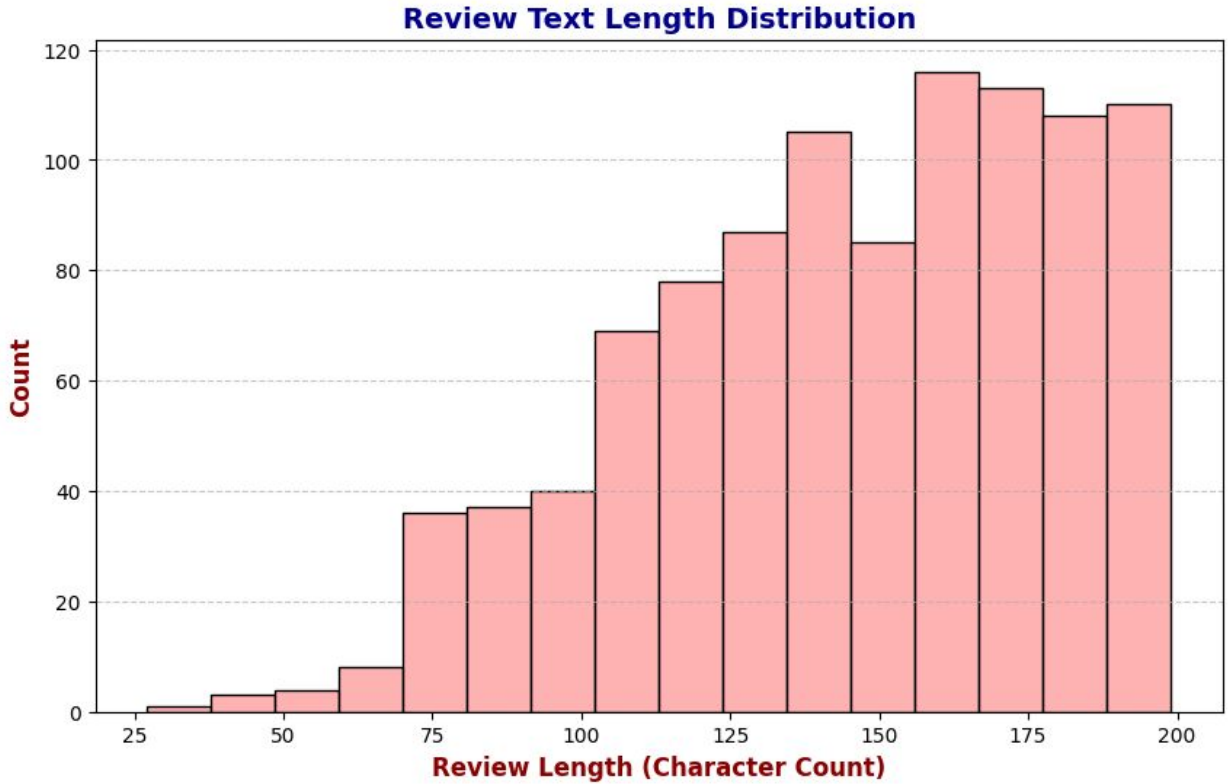
Slight Skew Towards Higher Ratings:

There's a slight tendency towards **higher ratings (3.5 to 4.5)**, suggesting a **generally positive experience**.

Presence of All Ratings:

All rating values are represented, indicating a wide range of customer experiences.

5) Customer Reviews Frequency



Right Skewed Distribution:

The distribution is heavily right-skewed, indicating **most reviews have a higher character count.**

Peak in Longer Reviews:

The peak of the distribution is towards the right, suggesting a **common review length in the 150-200 character range.**

Few Short Reviews:

There are relatively few reviews with very short character counts (below 75 characters).

Variability in Length:

There's a wide range of review lengths, from very short to relatively long.

Potential for Detailed Feedback:

The **prevalence of longer reviews suggests customers are willing to provide detailed feedback**.

SQL

Connect SQL

```python
import sqlite3
import pandas as pd

# File name of the uploaded CSV
csv_filename = "/content/df2-customer_feedback_sentiment_dataset.csv"  # Replace this with the uploaded file name

# Load CSV into a pandas DataFrame
df = pd.read_csv(csv_filename)

# Connect to SQLite database (or create a new one)
conn = sqlite3.connect("example.db")
cursor = conn.cursor()

# Write DataFrame to SQLite table
table_name = "df2"  # Specify your table name
df.to_sql(table_name, conn, if_exists="replace", index=False)

print(f"Table '{table_name}' created in SQLite database.")

```

```
Table 'df2' created in SQLite database.
```

1) Retrieve all reviews:

```
SELECT * from df2;
```

Review_ID	Customer_ID	Review_Text	Sentiment_Label	Rating	Service_Type
bdd640fb-0667-4ad1-9c80-317fa3b1799d	23b8c1e9-3924-46de-beb1-3b9046685257	Beautiful instead ahead despite measure ago cu...	Neutral	1	Claim
12476f57-a5e5-45ab-aefc-fad8efc89849	88bd6407-2bcf-4e01-a28d-efe39bf00273	Left establish understand read. Range successf...	Neutral	3	Claim
cac5b68c-28f4-4481-a0a0-4dc427209bdf	10435a10-98ae-4334-ac12-ace8ae340454	Other life edge network wall quite. Race Mr en...	Positive	2	Customer Support
913e4de2-e0c5-4cb8-bda9-c2a90ed42f1a	bb5e4bcf-15ed-4269-9429-6c07f26b4776	Within mouth call process. Close month parent ...	Positive	5	Claim
dfde4fbf-3ff3-40bf-b66e-cb15474ebc19	ceda8bbb-7171-4434-934c-6c92ec5b227c	Anything yourself structure why. Coach magazin...	Neutral	4	Claim

2) Count total reviews:

```
SELECT count(Review_Text) from df2;
```

	count(Review_Text)
0	1000

3) Find unique sentiment labels:

```
SELECT DISTINCT Sentiment_Label from df2;
```

	Sentiment_Label
0	Neutral
1	Positive
2	Negative

4) List all unique service types:

```
SELECT DISTINCT Service_Type from df2;
```

	Service_Type
0	Claim
1	Customer Support
2	Policy Purchase

5) Find all reviews with a rating of 5:

```
SELECT Review_Text,Rating from df2 where
Rating=5;
```

	Review_Text	Rating
cell output actions		
	...outh call process. Close month parent ...	5
1	About side PM. Claim kind relationship night b...	5
2	Voice boy wife condition while enter. Others g...	5
3	Soldier where save probably exist professional...	5
4	Ahead picture son report. Nearly need behavior...	5
...	...	...
183	Past lose window do place card often mission. ...	5
184	Apply house heart heavy student suffer. Ahead ...	5
185	Use board despite task economic partner work d...	5
186	Thus night strong Congress high. Member finall...	5
187	Who goal majority quite. Smile information sur...	5

188 rows × 2 columns

188 Reviews have 5 Ratings Out of 1000 Reviews

6) Find all reviews with negative sentiment (assuming Rating = 1 is negative)

```
SELECT Review_Text,Sentiment_Label,Rating from df2 where
Sentiment_Label='Negative' and Rating=1;
```

	Review_Text	Sentiment_Label	Rating
0	Major event magazine home protect. Right subje...	Negative	1
1	Bill activity expect long future whole educati...	Negative	1
2	Buy read record wall matter management. Our th...	Negative	1
3	Discover top realize. Little another avoid und...	Negative	1
4	Buy tax kid. Yeah into yet. Think few themselv...	Negative	1
...	...	...	...
68	Form both cause professional. Season day senio...	Negative	1
69	Firm along church every school too. Such profe...	Negative	1
70	Interview view offer purpose affect view. Inte...	Negative	1
71	Particular will imagine civil. Get national pr...	Negative	1
72	Grow evidence tax prepare force simple my tax....	Negative	1

73 rows × 3 columns

73 Reviews has Rating is 1 out of 1000 Datasets

7) Find reviews from a specific customer (e.g., Customer_ID = 100):

```
SELECT Review_Text from df2 where
Customer_ID='b945f8ed-b694-494b-9b8c-d3a442ffd324';
```

	Review_Text
0	Pass memory stuff look decision agreement mana...

8) Count how many reviews each sentiment label has:

```
SELECT Sentiment_Label,count(Review_Text)
from df2 GROUP BY Sentiment_Label;
```

	Sentiment_Label	count(Review_Text)
0	Negative	331
1	Neutral	354
2	Positive	315

9) Find the average rating for each service type:

```
SELECT Service_Type,avg(Rating) as
AVG_Rating from df2 GROUP BY Service_Type
Order By AVG_Rating DESC;
```

	Service_Type	AVG_Rating
0	Policy Purchase	3.119403
1	Claim	2.975460
2	Customer Support	2.938053

10) Find the highest-rated service type:

```
SELECT Service_Type,count(Rating) as count_Rating from
df2 where Rating='5' GROUP BY Service_Type order by
count_Rating DESC;
```

	Service_Type	count_Rating
0	Policy Purchase	69
1	Claim	63
2	Customer Support	56

```
SELECT Service_Type,count(Rating) as count_Rating from
df2 where Rating='1' GROUP BY Service_Type order by
count_Rating DESC;
```

	Service_Type	count_Rating
0	Customer Support	69
1	Claim	65
2	Policy Purchase	64

11) Find the number of reviews per customer:

```
SELECT Customer_ID,count(Review_Text) from df2 GROUP BY Customer_ID;
```

	Customer_ID	count(Review_Text)
0	008b8631-ad4f-4658-8fc6-7d4c62a7ce6f	1
1	00926d61-368b-4d68-83c6-b5324b25c483	1
2	00adad8f-cc5e-479b-a7fc-50e01e7e0e2d	1
3	00c4f5d9-cf96-45ca-99b9-49214a6575ee	1
4	00f20f74-3393-4094-a8b2-0e34b8bd0a51	1
...	...	...
995	fea82e3a-b544-41cb-9541-0a9d090aaa93	1
996	feb08155-c435-40eb-9753-c87fe50c3919	1
997	fecba0a6-ac2c-4e6a-8826-449f00bf94e3	1
998	fed024cb-b6f3-4bd4-8409-b7c8ef59a4b5	1
999	ff99fe31-1006-40eb-a3a8-4422e29eae63	1

1000 rows × 2 columns

All Reviews are Unique

12) Find customers who have submitted more than 5 reviews:

```
SELECT Customer_ID,Rating from df2 Where Rating=5;
```

	Customer_ID	Rating
0	bb5e4bcf-15ed-4269-9429-6c07f26b4776	5
1	6c6f7633-a260-4723-97a0-df490d01280f	5
2	6651529e-8268-490b-a438-25b559e4b671	5
3	36b5229a-acf5-481e-b131-62697118e364	5
4	b2c08394-e17f-49e1-b028-604649bc473f	5
...	...	...
183	7dbc5a27-0982-46a4-9526-a8453fdd92d8	5
184	c7b1784d-e868-433b-a3ba-7bccd70dbe88	5
185	7334602a-7c26-4f12-bc0b-ba0c22376ea7	5
186	481527d4-4689-43bb-ba87-b4bca9a72234	5
187	3fb79c50-b072-4aed-9772-f741f0301cc9	5

188 rows × 2 columns

Total - 188 Users Gives Rating 5 Out of 1000 Users

Preprocessing

1) Missing Values

```
1 df2.isnull().sum()
```

	0
Review_ID	0
Customer_ID	0
Review_Text	0
Sentiment_Label	0
Rating	0
Service_Type	0

dtype: int64

No Missing Values

Review_ID	Customer_ID	Review_Text	Sentiment_Label	Rating	Service_Type
bdd640fb-0667-4ad1-9c80-317fa3b1799d	23b8c1e9-3924-46de-beb1-3b9046685257	Quality throughout beautiful instead ahead des...	Negative	2	Claim
c241330b-01a9-471f-9e8a-774bcf36d58b	6c307511-b2b9-437a-a8df-6ec4ce4a2bbd	Everyone democratic shake bill here grow gas e...	Negative	1	Claim
8d5288f1-142c-4fe8-a0e7-a113ec1b8ca1	9e574f7a-a0ee-49ae-9453-dd324b0dbb41	Officer relate animal direction eye bag do big...	Negative	1	Customer Support
451b4cf3-6123-4df7-b656-af7229d4beef	b02b61c4-a3d7-4628-ace6-6fa2fd5166e6	Detail food shoulder argue start source husban...	Positive	4	Policy Purchase
ff5e9ff0-ff50-4de4-b825-67b85cabcc97	1745d6d8-7e57-4ddf-8270-50a82369b584	Technology check civil quite others.	Negative	1	Claim

1) Sentiment Labeling

```
# Update Sentiment_Label based on Rating
df2['Sentiment_Label'] = df2['Rating'].apply(lambda x: 'Negative' if x in [1, 2]
                                             else 'Neutral' if x == 3
                                             else 'Positive' if x in [4, 5]
                                             else None)

```

Review_Text	Sentiment_Label	Rating	Service_Type
Quality throughout beautiful instead ahead des...	Negative	2	Claim
Everyone democratic shake bill here grow gas e...	Negative	1	Claim
Officer relate animal direction eye bag do big...	Negative	1	Customer Support
Detail food shoulder argue start source husban...	Positive	4	Policy Purchase
Technology check civil quite others.	Negative	1	Claim

2) Text Cleaning & Tokenization

i) Taken Review_Text Column for Text Classification

```
df2['Review_Text'].head()
```

	Review_Text
0	Quality throughout beautiful instead ahead des...
1	Everyone democratic shake bill here grow gas e...
2	Officer relate animal direction eye bag do big...
3	Detail food shoulder argue start source husban...
4	Technology check civil quite others.

dtype: object

ii) Stopwords Removal for df2['Review_Text']

NLTK (Natural Language Toolkit):

NLTK (Natural Language Toolkit):

```
]    1 # Step 1: Uninstall NLTK and delete its data entirely
     2 !pip uninstall -y nltk
     3 !rm -rf /root/nltk_data
     4 !rm -rf /usr/local/nltk_data
     5
     6 # Step 2: Reinstall a stable version of NLTK
     7 !pip install nltk==3.8.1  # Pin to a stable version
```

```
Found existing installation: nltk 3.9.1
Uninstalling nltk-3.9.1:
  Successfully uninstalled nltk-3.9.1
Collecting nltk==3.8.1
  Downloading nltk-3.8.1-py3-none-any.whl.metadata (2.8 kB)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk==3.8.1) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk==3.8.1) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk==3.8.1) (2024.1
```

Using NlTK Library

```
from nltk.corpus import stopwords
```

```
# Get the list of stopwords
stop_words = set(stopwords.words('english'))

# Function to remove stopwords
def remove_stopwords(text):
    words = word_tokenize(text.lower())  # Tokenize and convert to lowercase
    filtered_words = [word for word in words if word.isalnum() and word not in stop_words]
    return ' '.join(filtered_words)

# Apply function to 'Review_Text' column
df2['Cleaned_Review'] = df2['Review_Text'].apply(remove_stopwords)

print(df2)
```

Stop Words: The, "And, "Is, "A, "An, "About, "Each, and "Such.

```
                                          Cleaned_Review
0    quality throughout beautiful instead ahead des...
1    everyone democratic shake bill grow gas enough...
2    officer relate animal direction eye bag big ev...
3    detail food shoulder argue start source husban...
4                  technology check civil quite others
..                                                 ...
995  speak condition cover admit real animal clear ...
996  everyone real benefit claim recognize book nea...
997  network class individual special color sing ar...
998  value tax concern new threat think maybe throw...
999  tree seek computer lawyer easy need whole boar...

[1000 rows x 7 columns]
```

Why remove stop words?

- **To improve the accuracy and efficiency of NLP tasks**
- **To focus on more meaningful words**
- **To reduce the size of the dataset and the time it takes to train models**

iii)Remove Punctuation, and Special Characters.

iv) Tokenization

Sentence, Word, Letter Tokenization

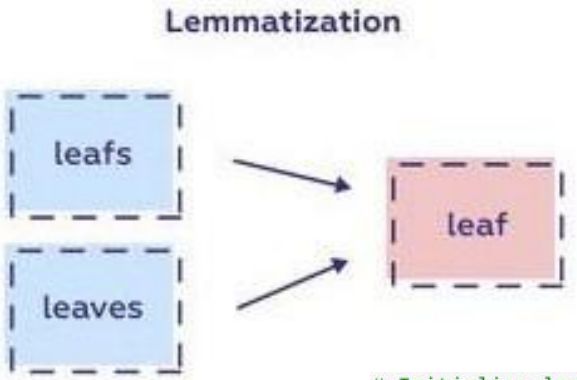
Tokenized_Review
[quality, beautiful, instead, ahead, despite, ...]
[democratic, shake, grow, gas, analysis]

```
1
2 # Tokenization function (split by spaces)
3 def tokenize(text):
4     return text.split()
5
6 # Apply tokenization
7 df2['Tokenized_Review'] = df2['Remove_Punchuation_Review'].apply(tokenize)
8
9 print(df2)
10
```

v) Lemmatize text

```
from nltk.stem import WordNetLemmatizer
import nltk

# Download necessary NLTK data (run these once)
nltk.download('wordnet')
nltk.download('omw-1.4')
```




```
# Initialize lemmatizer
lemmatizer = WordNetLemmatizer()

# Lemmatization function
def lemmatize_text(text):
    if isinstance(text, list):        # If text is already a list of words
        words = text
    else:                             # If it's a string, split into words
        words = str(text).split()

    lemmatized_words = [lemmatizer.lemmatize(word.lower()) for word in words]  # Lowercase and lemmatize
    return ' '.join(lemmatized_words)

# Apply lemmatization
df2['Lemmatized_Review'] = df2['Tokenized_Review'].apply(lemmatize_text)
```

Review_Test Column was done by

1) Stopwords Removal,

2) Punchuation and Special Character Removal,

3) Done by Tokenization,

4) Done by Lemmatization.

all process done

final process was Lemmatization **so Kept only Lemmatized_Review Column** else delete Columns Like

[Review_Text, Cleaned_Review, Remove_Punchuation_Review, Tokenized_Review]

```
[ ]  1 df2.drop(['Review_Text', 'Cleaned_Review', 'Remove_Punchuation_Review', 'Tokenized_Review'], axis=1, inplace=True)
```

1) Feature Extraction

Vectorization

1) TF-IDF, Word2Vec, BERT embeddings

TF-IDF: Vectorizes text based on term importance.

Word2Vec: Learns word embeddings; we average them to represent the sentence.

BERT: Contextual embeddings for entire sentences using a pre-trained BERT model

How TF-IDF works

- **Term frequency (TF)**: Counts how many times a word appears in a document
- **Inverse document frequency (IDF)**: Measures how common or rare a word is across a collection of documents
- **TF-IDF**: Calculates a score for each word by multiplying TF and IDF

Why TF-IDF is useful

- It helps machine learning models read words
- It's used in text classification
- It's used to rank the relevance of documents in search engines
- It's used to improve the quality of text generation in large language models (LLMs)

	Lemmatized_Review	ability	able	accept	\
0	quality beautiful instead ahead despite measur...	0.0	0.0	0.0	
1	democratic shake grow gas analysis	0.0	0.0	0.0	
2	officer relate animal direction eye bag big pl...	0.0	0.0	0.0	
3	food shoulder argue start source husband tree ...	0.0	0.0	0.0	
4	technology check civil quite	0.0	0.0	0.0	

	according	...	world	worry	write	writer	wrong	yard	yeah	year	yes	\
0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	young
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

TF-IDF vs. bag-of-words

TF-IDF is an improvement on the bag-of-words model because it considers how common or rare a word is across a collection of documents.

TF-IDF challenges

TF-IDF can be sensitive to stop words, which are common words that don't carry much meaning. **Removing stop words can help with this issue.**

Word2Vec

Word2Vec Exact of (Word to Vectorization) - words are embedding [Encoding] to understand Machine to train models

- Word2Vec can be used to detect synonymous words
- It can suggest additional words for a partial sentence
- It can be used in language translation
- It can be used in text categorization

```
df2_cleaned['Word2Vec_Embedding'].head()
```

	Word2Vec_Embedding
0	[-0.005997389, 0.014081386, 0.0038841476, 0.00...
1	[-0.0050376747, 0.0060141226, 0.0013677565, 0....
2	[-0.0031995953, 0.008657489, 0.002409692, 0.00...
3	[-0.0055310912, 0.010755829, 0.0058071865, 0.0...
4	[-0.007638862, 0.008539527, 0.00551427, -0.001...

dtype: object

BERT Embedding

The BERT model transforms input text into high-dimensional vectors (embeddings) that represent the semantic meaning of the text

Fine-tuning:

To adapt BERT for specific tasks like sentiment analysis or question answering, you can further fine-tune the model on relevant data.

BERT Embedding

from transformers import BertTokenizer, BertModel

```
# 3. BERT Embeddings
bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embedding(text):
    inputs = bert_tokenizer(text, return_tensors='pt', padding=True, truncation=True)
    outputs = bert_model(**inputs)
    last_hidden_state = outputs.last_hidden_state
    return list(last_hidden_state.mean(dim=1).detach().numpy()[0])

df2_cleaned['BERT_Embedding'] = df2_cleaned['Lemmatized_Review'].apply(get_bert_embedding)

```

1. Import the BERT tokenizer and model from the Hugging Face Transformers library

2. Load the pre-trained BERT tokenizer ('bert-base-uncased' means it's a base model with lowercase text)

3. Load the pre-trained BERT model ('bert-base-uncased' matches the tokenizer)

4. Define a function to get the BERT embedding for a given text

    # 5. Tokenize the input text and convert it into the format BERT expects (PyTorch tensors)

    # 6. Pass the tokenized input into the BERT model and get its output

    # 7. Extract the last hidden state (the contextualized word embeddings)

    # 8. Average the embeddings across all tokens to get a single sentence-level embedding

```
1 df2_cleaned['BERT_Embedding'].head()
```

	BERT_Embedding
0	[-0.006879185, -0.17997745, 0.28093535, -0.055...
1	[-0.090950586, -0.3018164, -0.15380336, 0.2520...
2	[-0.010879632, -0.1772102, 0.20189424, 0.01788...
3	[-0.092359215, 0.044332672, 0.18089476, 0.0346...
4	[0.39030206, -0.21246122, -0.15707244, 0.08479...

dtype: object

BERT embeddings is best for model training so i am taking that result

Bidirectional Encoder Representations from Transformers embeddings

```
[ ]  1 print(df2_cleaned['BERT_Embedding'].dtype)
     2 print(df2_cleaned['BERT_Embedding'].head())
```

```
object
0    [-0.006879185, -0.17997745, 0.28093535, -0.055...
1    [-0.090950586, -0.3018164, -0.15380336, 0.2520...
2    [-0.010879632, -0.1772102, 0.20189424, 0.01788...
3    [-0.092359215, 0.044332672, 0.18089476, 0.0346...
4    [0.39030206, -0.21246122, -0.15707244, 0.08479...
Name: BERT_Embedding, dtype: object
```

```
[ ]  1 print(type(df2_cleaned['BERT_Embedding'].iloc[0]))  # Check the type of one row
```

```
<class 'list'>
```

```python
from sklearn.decomposition import TruncatedSVD
import pandas as pd

# Convert BERT embeddings (list of lists) into separate columns
bert_df = pd.DataFrame(df2_cleaned['BERT_Embedding'].tolist(), index=df2_cleaned.index)

# Reduce dimensions with Truncated SVD
svd = TruncatedSVD(n_components=1)  # Reduce to one dimension
bert_reduced = svd.fit_transform(bert_df)

# Add reduced BERT embeddings back to the main DataFrame as a single column
df2_cleaned['BERT_Embedding'] = bert_reduced.flatten()

# Optional: drop high-dimensional columns if needed
print(df2_cleaned.shape)
print(df2_cleaned.head())

```

- `TruncatedSVD` from `sklearn.decomposition`: This is the algorithm you're using for dimensionality reduction.
- `pandas` as `pd`: The library you're using to handle your DataFrame.

```
bert_df = pd.DataFrame(df2_cleaned['BERT_Embedding'].tolist(), index=df2_cleaned.index)
```

- `df2_cleaned['BERT_Embedding']` contains BERT embeddings — each row is likely a list of numbers (a high-dimensional vector).
- `.tolist()` converts that column of vectors into a list of lists.
- `pd.DataFrame()` creates a new DataFrame, `bert_df`, whe ch number in the BERT embedding becomes its own column. So if each embedding is 768 dimensions, you'd get 768 separate columns.
- `index=df2_cleaned.index` keeps the row alignment between `bert_df` and `df2_cleaned`.

```
6
7 # Reduce dimensions with Truncated SVD
8 svd = TruncatedSVD(n_components=1)  # Reduce to one dimension
9 bert_reduced = svd.fit_transform(bert_df)
10
```

- `TruncatedSVD(n_components=1)` creates an SVD model that reduces your embeddings to just **one dimension** (a single number for each row).
- `fit_transform(bert_df)` fits the SVD model on your high-dimensional BERT embeddings and transforms them into this new 1D representation.
- `bert_reduced` is now a 2D array where each row looks like `[value]`.

```
# Add reduced BERT embeddings back to the main DataFrame as a single column
df2_cleaned['BERT_Embedding'] = bert_reduced.flatten()
```

- `bert_reduced.flatten()` turns the 2D array into a 1D array, so each row just has one number.
- `df2_cleaned['BERT_Embedding']` gets overwritten — the original high-dimensional vector is replaced with its single reduced dimension.

Review_ID	Customer_ID	Sentiment_Label	Rating	Service_Type	Lemmatized_Review	Word2Vec_Embedding	BERT_Embedding
bdd640fb-0667-4ad1-9c80-317fa3b1799d	23b8c1e9-3924-46de-beb1-3b9046685257	Negative	2	Claim	quality beautiful instead ahead despite measur...	[-0.0061829146, 0.013488408, 0.0035640725, 0.0...	-7.066508
c241330b-01a9-471f-9e8a-774bcf36d58b	6c307511-b2b9-437a-a8df-6ec4ce4a2bbd	Negative	1	Claim	democratic shake grow gas analysis	[-0.0054913587, 0.0057755834, 0.0012450566, 0....	-6.115204
8d5288f1-142c-4fe8-a0e7-a113ec1b8ca1	9e574f7a-a0ee-49ae-9453-dd324b0dbb41	Negative	1	Customer Support	officer relate animal direction eye bag big pl...	[-0.0035938728, 0.00843976, 0.0021786473, 0.00...	-7.651156
451b4cf3-6123-4df7-b656-af7229d4beef	b02b61c4-a3d7-4628-ace6-6fa2fd5166e6	Positive	4	Policy Purchase	food shoulder argue start source husband tree ...	[-0.0059934086, 0.010755725, 0.005541168, 0.00...	-7.367544
ff5e9ff0-ff50-4de4-b825-67b85cabcc97	1745d6d8-7e57-4ddf-8270-50a82369b584	Negative	1	Claim	technology check civil quite	[-0.008219169, 0.008976587, 0.005485099, -0.00...	-5.620431

Feature Selection

```
feature = df2[['BERT_Embedding','Rating','Service_Type_Encoded']]
target = df2['Sentiment_Label']
```

Bert_Ebedding is From Review Text -> Why am using Rating and Service_Type Compare with Review Text

Reason:

Machine could understand what about the words related to Positive and which Words Related Negative so if am writing -

Eg:

“This is Best Product” this words equal to the Rating of 4 or 5 that target is Positive, so Machine can Understand Positive Related All Words

```
1 df2['Sentiment_Label'].value_counts()
```

Sentiment_Label	count
Positive	388
Neutral	359
Negative	253

dtype: int64

Target is Multiclass

Machine Learning

```python
# Now your features are properly shaped for RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# RandomForest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Predictions
y_pred = rf_classifier.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

```

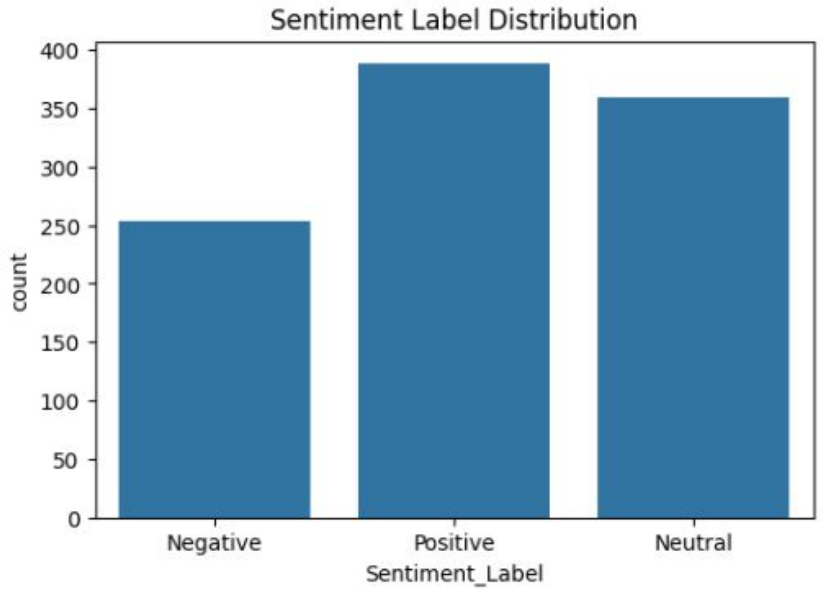
```
Accuracy: 0.8
Classification Report:
              precision    recall  f1-score   support

    Negative       0.86      0.83      0.85        53
     Neutral       0.71      0.71      0.71        69
    Positive       0.84      0.86      0.85        78

    accuracy                           0.80       200
   macro avg       0.80      0.80      0.80       200
weighted avg       0.80      0.80      0.80       200
```

Check Dataset is Balanced or Unbalanced

```
Sentiment_Label
Positive    388
Neutral     359
Negative    253
Name: count, dtype: int64
```




```
Dataset is IMBALANCED
```

Dataset is Imbalanced so using Smote method

```
Before SMOTE: Counter({'Positive': 310, 'Neutral': 290, 'Negative': 200})
After SMOTE: Counter({'Positive': 310, 'Neutral': 310, 'Negative': 310})
```

If Dataset is imbalanced - Machine will Learn Data is Biased or Sometimes Data Leakage also happens that means Machine will Memorize majority class so - using Cross Validation also to Machine Learn Without or Low Bias or

Smote Method - [**Synthetic Minority Oversampling Technique] - Like Oversampling with Minority Class filled with Artificial Data for machine Learning without or Low Bias**

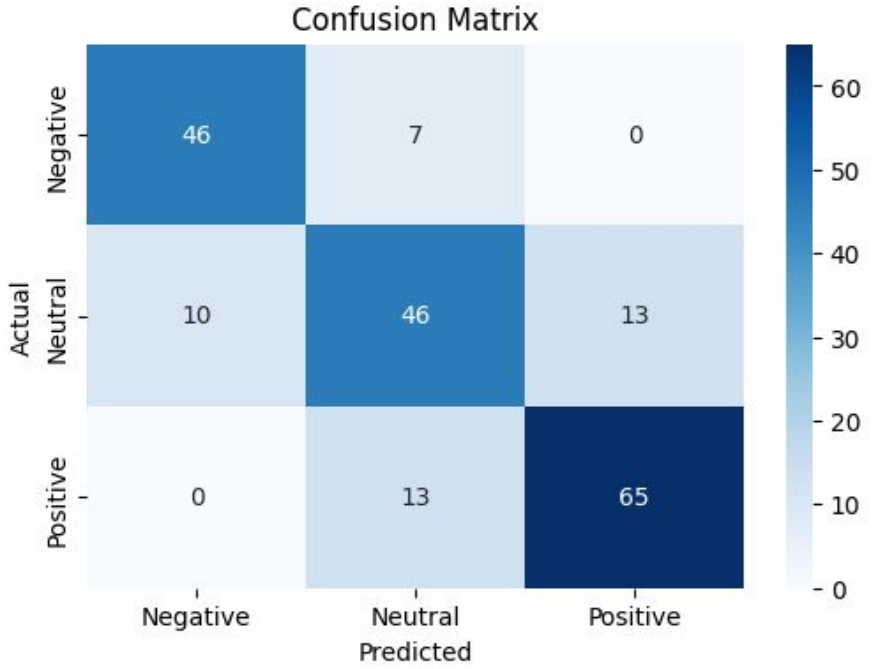
After [SMOTE]

Train Accuracy: 1.00

Test Accuracy: 0.79

🚨 Model is OVERFITTING: High training accuracy, lower test accuracy

Random Forest Classifier



Finding Best Params for Random Forest using Hyperparameter Tuning

Finding Best Parameter for My Model

```
# Hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],        # Number of trees
    'max_depth': [10, 20, None],            # Depth of each tree
    'min_samples_split': [2, 5, 10],        # Min samples needed to split
    'min_samples_leaf': [1, 3, 5],         # Min samples per leaf
    'max_features': ['sqrt', 'log2'],       # Number of features to consider
    'criterion': ['gini', 'entropy']        # Splitting criterion
}
```

Grid Search CV runs each 5 folds for finding best params

```
# Random Forest model
rf = RandomForestClassifier(random_state=42)

# GridSearchCV
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,                       # 5-fold cross-validation
    scoring='accuracy',         # Optimize for accuracy
    n_jobs=-1,                  # Use all available CPUs
    verbose=2                   # Show progress
)
```

Fitting 5 folds for each of 324 candidates, totalling 1620 fits

Best Parameters: {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}

```
Train Accuracy: 0.97

Test Accuracy: 0.80
```

🚨 Model is OVERFITTING: High training accuracy, lower test accuracy.

Still Model is Overfit but little bit reduce train accuracy after of Hyperparameter Tuning

2) XGBoosting Algorithm - Train Balanced Dataset after - SMOTE

Train Accuracy: 0.95

Test Accuracy: 0.80

🚨 `Model is OVERFITTING: High training accuracy, lower test accuracy.`

Little bit vary compare to Random Forest Algorithm

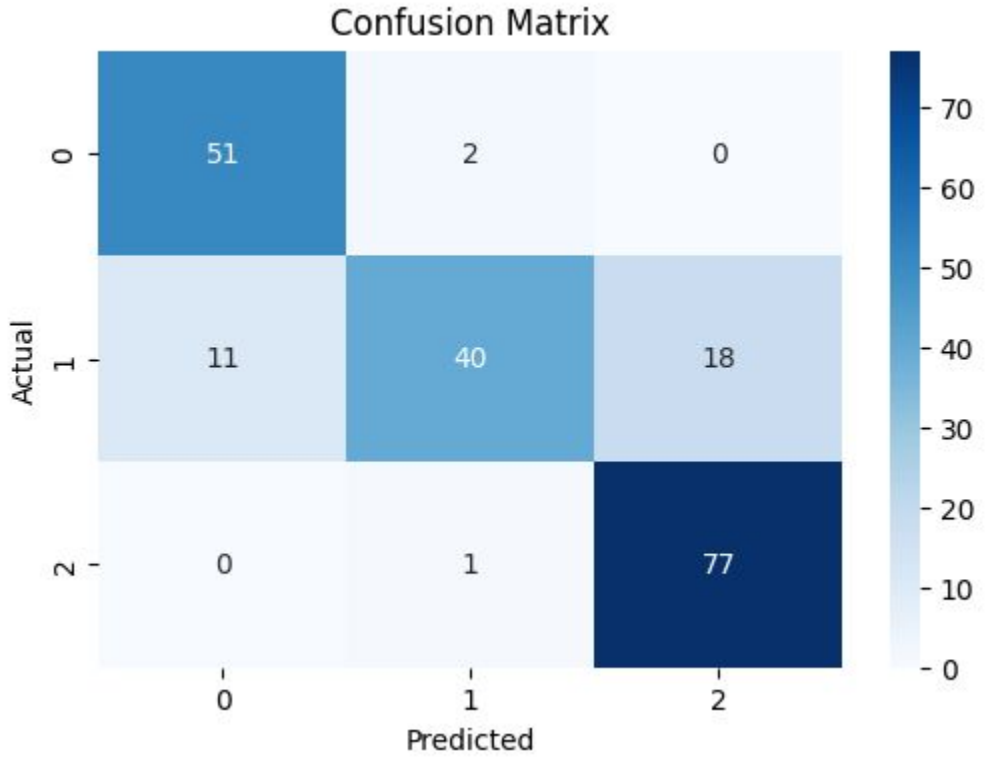
3) **Navie_bayes Algorithm**

```
Train Accuracy: 0.85
Test Accuracy: 0.84

Classification Report (Test Set):
              precision    recall  f1-score   support

           0       0.82      0.96      0.89        53
           1       0.93      0.58      0.71        69
           2       0.81      0.99      0.89        78

    accuracy                           0.84       200
   macro avg       0.85      0.84      0.83       200
weighted avg       0.86      0.84      0.83       200
```



Train Accuracy: 0.86

Test Accuracy: 0.84

Model is Genralized Well Overfit is Reduced Compare to XGboost and Random Forest

Prediction

```python
import joblib

# Save the model
joblib.dump(nb_classifier, 'naive_bayes_model.pkl')

print("Model saved successfully!")

```

```
Model saved successfully!
```

```python
import joblib
import pandas as pd

# Load the saved Naive Bayes model
nb_classifier_loaded = joblib.load('naive_bayes_model.pkl')
print("Naive Bayes model loaded successfully!")

# Example: New data for prediction (ensure it's preprocessed the same way)
new_data = pd.DataFrame({
    'BERT_Embedding': [-7.066508],
    'Rating': [2],  # Example rating value, adjust as needed
    'Service_Type_Encoded': [0]  # Example encoded service type, adjust based on your encoding
})


# Predict class probabilities
class_probabilities = nb_classifier_loaded.predict_proba(new_data)

# Predicted class (0 or 1)
predicted_class = nb_classifier_loaded.predict(new_data)

print(f"Predicted class: {predicted_class[0]}")
print(f"Class probabilities: {class_probabilities}")

```

```
Naive Bayes model loaded successfully!
Predicted class: 0
Class probabilities: [[7.58270737e-01 2.41728838e-01 4.25096567e-07]]
```

```python
import joblib
import pandas as pd

# Load the saved Naive Bayes model
nb_classifier_loaded = joblib.load('naive_bayes_model.pkl')
print("Naive Bayes model loaded successfully!")

# Example: New data for prediction (ensure it's preprocessed the same way)
new_data = pd.DataFrame({
    'BERT_Embedding': [-7.367544],
    'Rating': [4],  # Example rating value, adjust as needed
    'Service_Type_Encoded': [2]  # Example encoded service type, adjust based on your encoding
})


# Predict class probabilities
class_probabilities = nb_classifier_loaded.predict_proba(new_data)

# Predicted class (0 or 1)
predicted_class = nb_classifier_loaded.predict(new_data)

print(f"Predicted class: {predicted_class[0]}")
print(f"Class probabilities: {class_probabilities}")

```

```
Naive Bayes model loaded successfully!
Predicted class: 2
Class probabilities: [[2.90890783e-07 2.00045190e-01 7.99954520e-01]]
```

BERT Embedding to Sentence Format

- **Now Decodeling Method**

```
from sentence_transformers import SentenceTransformer
```

```
]
```

```python
import joblib
import pandas as pd
from sentence_transformers import SentenceTransformer

# Load the saved Naive Bayes model
nb_classifier_loaded = joblib.load('naive_bayes_model.pkl')
print("Naive Bayes model loaded successfully!")

# Load a pre-trained BERT model for sentence embeddings
bert_model = SentenceTransformer('all-MiniLM-L6-v2')
print("BERT model loaded successfully!")

# Function to convert user input sentence to BERT embedding
def get_bert_embedding(sentence):
    embedding = bert_model.encode(sentence)
    return embedding

# Get user input
user_sentence = 'Push dog build three east organization people information on mission various.'
bert_embedding = get_bert_embedding(user_sentence)

# Reduce BERT embedding to a single value (e.g., mean)
bert_embedding_reduced = bert_embedding.mean()  # Or use sum(), max(), etc.
```

```python
rating = 4
service_type_encoded = 2

# Create DataFrame
new_data = pd.DataFrame({
    'BERT_Embedding': [bert_embedding_reduced],  # Single numeric value
    'Rating': [rating],
    'Service_Type_Encoded': [service_type_encoded]
})

# Predict class probabilities and class
class_probabilities = nb_classifier_loaded.predict_proba(new_data)
predicted_class = nb_classifier_loaded.predict(new_data)[0]

# Map predicted class to review type
class_labels = {
    0: "Negative review",
    1: "Neutral review",
    2: "Positive review"
}

# Display the result
print(f"\nPredicted class: {predicted_class} ({class_labels[predicted_class]})")
print(f"Class probabilities: {class_probabilities}")

```

```
Naive Bayes model loaded successfully!
BERT model loaded successfully!

Predicted class: 1 (Neutral review)
```

Prediction is Failure

So Now am Using SVD to transform a BERT Embedding

```
[ ]  1 import joblib
     2
     3 # Load the saved SVD model
     4 svd = joblib.load('svd_model.pkl')
     5 print("SVD model loaded successfully!")
```

```
SVD model loaded successfully!
```

```python
import joblib
import pandas as pd
from sentence_transformers import SentenceTransformer
from sklearn.decomposition import TruncatedSVD

# Load the saved Naive Bayes model
nb_classifier_loaded = joblib.load('naive_bayes_model.pkl')
print("Naive Bayes model loaded successfully!")

# Load a pre-trained BERT model for sentence embeddings
bert_model = SentenceTransformer('bert-base-uncased')
print("BERT model loaded successfully!")

# Load the same SVD model used for dimensionality reduction
svd = joblib.load('svd_model.pkl')
print("SVD model loaded successfully!")

# Function to convert user input sentence to BERT embedding
def get_bert_embedding(sentence):
    embedding = bert_model.encode(sentence)
    return embedding

# Get user input
user_sentence = 'this product is superb'
bert_embedding = get_bert_embedding(user_sentence)

# Convert BERT embedding to DataFrame (1 sample, 768 features)
bert_embedding_df = pd.DataFrame([bert_embedding])
```

```
# Convert BERT embedding to DataFrame (1 sample, 768 features)
bert_embedding_df = pd.DataFrame([bert_embedding])

# Reduce dimensions using SVD to a single feature
bert_embedding_reduced = svd.transform(bert_embedding_df)  # Should reduce to one value

# Example additional features
rating = 3
service_type_encoded = 0
```

```
# Create DataFrame for prediction
new_data = pd.DataFrame({
    'BERT_Embedding': [bert_embedding_reduced.flatten()[0]],  # Single value, wrapped in a list
    'Rating': [rating],
    'Service_Type_Encoded': [service_type_encoded]
})

# Ensure order matches model's expected features
new_data = new_data[nb_classifier_loaded.feature_names_in_]
```

```python
# Ensure order matches model's expected features
new_data = new_data[nb_classifier_loaded.feature_names_in_]

# Predict class probabilities and class
class_probabilities = nb_classifier_loaded.predict_proba(new_data)
predicted_class = nb_classifier_loaded.predict(new_data)[0]

# Map predicted class to review type
class_labels = {
    0: "Negative review",
    1: "Neutral review",
    2: "Positive review"
}

# Display the result
print(f"\nPredicted class: {predicted_class} ({class_labels[predicted_class]})")
print(f"Class probabilities: {class_probabilities}")

```

```
WARNING:sentence_transformers.SentenceTransformer:No sentence-transformers model found with name bert-base-uncased. Creating a new one with mean pooling.
Naive Bayes model loaded successfully!
BERT model loaded successfully!
SVD model loaded successfully!

Predicted class: 1 (Neutral review)
Class probabilities: [[0.00360301 0.99525008 0.00114691]]
```

Streamlit Application

Using Blob_Text Method for Sentiment Analysing

Reason:

Using Synthetic Dataset So Positive, Neutral and Negative Arranges Wrongly So prediction is Mismatched so using Text Blob Method

Another Reason using 3 Feature Columns for Model Training, Now Facing Issues

White 3 Feature in 1 Dimension - only Uses Test Box

```
%%writefile app1.py
import streamlit as st
import joblib
import pandas as pd
from sentence_transformers import SentenceTransformer
from textblob import TextBlob

# Load models
nb_classifier_loaded = joblib.load('naive_bayes_model.pkl')
svd = joblib.load('svd_model.pkl')
bert_model = SentenceTransformer('bert-base-uncased')

# Function to convert user input sentence to BERT embedding
def get_bert_embedding(sentence):
    embedding = bert_model.encode(sentence)
    return embedding

# Streamlit UI
st.title('Sentiment Classification App')

# User input
user_sentence = st.text_input('Enter a sentence for sentiment analysis:')

if st.button('Predict Sentiment'):
    if user_sentence:
        # Use TextBlob for sentiment analysis
        blob = TextBlob(user_sentence)
```

```python
# Streamlit UI
st.title('Sentiment Classification App')

# User input
user_sentence = st.text_input('Enter a sentence for sentiment analysis:')

if st.button('Predict Sentiment'):
    if user_sentence:
        # Use TextBlob for sentiment analysis
        blob = TextBlob(user_sentence)
        sentiment = blob.sentiment
        st.write("TextBlob Sentiment Polarity:", sentiment.polarity)
        st.write("TextBlob Sentiment Subjectivity:", sentiment.subjectivity)
        st.write("TextBlob Sentiment:", "Positive" if sentiment.polarity > 0 else "Negative" if sentiment.polarity < 0 else "Neutral")
    else:
        st.warning('Please enter a sentence for analysis.')

```

Sentiment Classification App

Enter a sentence for sentiment analysis:

I switched to the Galaxy S25 Ultra after using the iPhone 12 Pro Max for 4 years. The display is absolutel

Predict Sentiment

TextBlob Sentiment Polarity: `0.25861111111111107`

TextBlob Sentiment Subjectivity: `0.5958333333333333`

TextBlob Sentiment: Positive

Feature Selection / Model Training - 2

Using Previous Dataset Synthetic Based Get Accuracy Only - 36% after Training

1) Decision Tree Classifier
2) Random Forest
3) XGBoosting
4) KNN
5) Navie Bayes [Prevent Overfit] Accuracy increase to 51%

My Dataset was IMBalanced so Using SMOTE - Still Not Get Accuracy So Am using

6) Using Hyperparameter Tuning for Random Forest - Run 28 Minutes to Get Best Params - using GridSearchCV

Next Step - Neural Network - Deep Learning

1st Step - FNN -

FeedForward Neural Network (FNN) using MLP (MultiLayer Perceptron)

Fully Connected Layer

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Define neural network
model = Sequential([
    Dense(128, activation='relu', input_shape=(50,)),  # 50 components from SVD
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')  # 3 sentiment classes: Positive, Neutral, Negative
])

# Compile
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
```

1. **Import Libraries** – Import TensorFlow and Keras modules for building the neural network.

2. **Define Model** – Create a `Sequential` neural network with:

   - `Dense(128, activation='relu')` – First hidden layer with 128 neurons, ReLU activation, and input size of 50 (SVD-reduced BERT embeddings).
   - `Dropout(0.3)` – Drops 30% of neurons to prevent overfitting.
   - `Dense(64, activation='relu')` – Second hidden layer with 64 neurons and ReLU activation.
   - `Dropout(0.3)` – Another dropout layer for regularization.
   - `Dense(3, activation='softmax')` – Output layer with 3 neurons for sentiment classes (Positive, Neutral, Negative) using softmax for probabilities.

3. **Compile Model** – Uses Adam optimizer, sparse categorical cross-entropy loss (since `y_train` has integer labels), and tracks accuracy.
4. **Train Model** – Trains for 50 epochs with a batch size of 32, using `X_train, y_train`, and validates with `X_test, y_test`.

```
Epoch 49/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5698 - loss: 0.8942 - val_accuracy: 0.3300 - val_loss: 1.2040
Epoch 50/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6015 - loss: 0.8917 - val_accuracy: 0.3350 - val_loss: 1.2054
<keras.src.callbacks.history.History at 0x7b95eb806ed0>
```

Not Get Accuracy Model Get Overfit

Training Accuracy is 60% but Test Accuracy is 33%

Loss was reduced - Total 50 Epoch

NEXT am Using - RNN & LSTM

RNN - Recurrent Neural Network using Long Short Term Memory

LSTM - No Need SVD Decomposition - TruncatedSVD

```
[ ]  1 df2_raw_bert = pd.read_csv('df2_bert.csv')
     2 feature = df2_raw_bert['BERT_Embedding']
     3 target = df2_raw_bert['Sentiment_Label']
```

LSTM high Capable to Cover High Dimension Data

LSTM

How LSTM Decides What to Keep or Forget?

LSTMs have **three gates** that help manage information flow:

1. **Forget Gate** (🗑️ Trash Unimportant Info)
   - Decides what past information to **forget** based on its relevance.
   - If a piece of info is not useful anymore, it's discarded (like clearing old cache).
2. **Input Gate** (📝 Store Important Info)
   - Selects **new important information** to **add** to memory.
   - Updates the cell state with fresh, useful data.
3. **Output Gate** (📢 Send Useful Info Forward)
   - Determines what **information should be passed** to the next time step.
   - Ensures only **important and relevant** features are used in predictions.

- **Forget Gate** 🗑️ → Erases unimportant notes.
- **Input Gate** 📝 → Writes down key points from new info.
- **Output Gate** 📢 → Shares only the most relevant notes.

So, **LSTM "filters" the information**, keeping **long-term dependencies** while removing noise. 🚀

Convert to 3D -> LSTM Work as 3D

```
]    1 X = np.stack(df2_raw_bert['BERT_Embedding'].values)
     2
     3 # Expand dimension to fit LSTM input: (num_samples, sequence_length=1, embedding_dim)
     4 X = np.expand_dims(X, axis=1)
     5
     6 print("New X shape:", X.shape)  # Should be (num_samples, 1, embedding_dim)
```

```
New X shape: (1000, 1, 768)
```

🔹 LSTM's 3D Input Shape

LSTM models expect input data in the form of:

scss Copy code

```
(batch_size, time_steps, features)
```

Where:

- **batch_size** – Number of samples processed together.
- **time_steps** – The sequence length (how many time points per sample).
- **features** – Number of features per time step (e.g., word embedding size).

Yes! **Changing the sequence length (`time_steps`) can improve predictions**, especially when working with sequential data like sentences broken into word embeddings.

🔹 How Sequence Length Affects LSTM Predictions

1. **Shorter Sequences (e.g., `time_steps = 1`)**
   - Each sample is treated as **one time step** (like a single vector).
   - Works well for **sentence-level embeddings** (e.g., SVD-reduced BERT embeddings).
   - **Limitation**: Doesn't capture **word-level context** inside a sentence.
2. **Longer Sequences (e.g., `time_steps = 20, 50, 100`)**
   - Each sample now consists of **multiple time steps** (e.g., words in a sentence).
   - Useful for **raw BERT embeddings (word-level, `[seq_len, 768]`)**.
   - **Advantage**: LSTM can learn **contextual dependencies between words**.

```
model = Sequential([
    LSTM(128, activation='relu', input_shape=(seq_len, 768), return_sequences=True),
    Dropout(0.3),
    LSTM(64, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])
```

🔹 Does Increasing Sequence Length Improve Prediction?

✅ **YES, if your data is sequential** (e.g., word embeddings in a sentence).

❌ **NO, if you use sentence-level embeddings** (SVD-reduced BERT), because LSTM won't benefit from time dependencies.

🔹 Best Approach for Your Case

🔹 If you are using **SVD-reduced BERT embeddings (50D per sentence)** → **Keep** `time_steps = 1`, because there's no word-level sequence.

🔹 If you want to **increase prediction accuracy**, use **raw BERT embeddings with** `time_steps > 1` (word-level embeddings).

LSTM

```
import pandas as pd
import numpy as np
import ast
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split

# Load dataset
df2_raw_bert = pd.read_csv('df2_bert.csv')

# Convert embeddings from string to numpy arrays
df2_raw_bert['BERT_Embedding'] = df2_raw_bert['BERT_Embedding'].apply(lambda x:
np.array(ast.literal_eval(x)))

# Convert to proper numpy format
X = np.stack(df2_raw_bert['BERT_Embedding'].values)

# Fix shape for LSTM: Add extra dimension (sequence_length = 1)
X = np.expand_dims(X, axis=1)
```

```
# Convert text labels to integers
label_mapping = {'Negative': 0, 'Neutral': 1, 'Positive': 2}  # Adjust if needed
df2_raw_bert['Sentiment_Label'] = df2_raw_bert['Sentiment_Label'].map(label_mapping)

# Ensure labels are integers
y = df2_raw_bert['Sentiment_Label'].values.astype(np.int32)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state=42)

# Define LSTM Model
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.3),
    LSTM(64),
    Dropout(0.3),
    Dense(3, activation='softmax')
])

# Compile
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
```

2️⃣ Convert Embeddings from String to NumPy Arrays

```
df2_raw_bert['BERT_Embedding'] = df2_raw_bert['BERT_Embedding'].apply(lambda x: np.array(ast.literal_eval(x)))
```

- **Why is this needed?**
  - In CSV files, arrays are stored as strings (e.g., `"[0.1, 0.2, 0.3, ...]"`).
  - `ast.literal_eval(x)` converts the string **back into a list**.
  - `np.array(...)` ensures that each list becomes a NumPy array.

3️⃣ Convert to Proper NumPy Format

```python
X = np.stack(df2_raw_bert['BERT_Embedding'].values)
```

- Stacks all **BERT embeddings** from the DataFrame into a **2D NumPy array**.
- **Shape before expansion:**

```scss
(num_samples, embedding_dim)
```

  - `num_samples` = Number of sentences in the dataset.
  - `embedding_dim` = Size of each BERT embedding (e.g., 768 for full BERT).

4️⃣ Fix Shape for LSTM Input

```python
X = np.expand_dims(X, axis=1)
```

- LSTM **expects 3D input** → `(batch_size, time_steps, features)`.
- Expanding dimension makes `X` **(num_samples, 1, embedding_dim)`.
  - **1** is the **sequence length** (since each row is a sentence-level embedding).
  - Now **LSTM can process it correctly**.

5️⃣ Convert Sentiment Labels to Integers

python Copy code

```
label_mapping = {'Negative': 0, 'Neutral': 1, 'Positive': 2}
df2_raw_bert['Sentiment_Label'] = df2_raw_bert['Sentiment_Label'].map(label_mapping)
```

- Maps text labels (`'Negative'`, `'Neutral'`, `'Positive'`) to integers (`0, 1, 2`).
- **Ensures labels are numerical** for model training.

6️⃣ Convert Labels to NumPy Array

python

```
y = df2_raw_bert['Sentiment_Label'].values.astype(np.int32)
```

- Extracts labels as a NumPy array (`y`).
- Ensures data type is **int32** for classification.

1. **LSTM(128, return_sequences=True)**
   - First LSTM layer with **128 units**.
   - `return_sequences=True` → Outputs a sequence (needed for stacked LSTMs).
2. **Dropout(0.3)**
   - Drops **30%** of neurons to **prevent overfitting**.
3. **LSTM(64)**
   - Second LSTM layer with **64 units**.
   - Since this is the final LSTM, `return_sequences=False` (default).
4. **Dropout(0.3)**
   - Another **30% dropout** to regularize.
5. **Dense(3, activation='softmax')**
   - Fully connected layer with **3 output neurons** (for `Negative, Neutral, Positive` classes).
   - Uses **softmax** activation for **multi-class classification**.

9 Compile Model

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

- **Optimizer**: `adam` → Adaptive learning rate for better convergence.
- **Loss function**: `sparse_categorical_crossentropy`
  - Used for **multi-class classification** when labels are integers (`0,1,2`).
- **Metric**: `accuracy` → To evaluate performance.

🔟 Train the Model

```
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
```

- **Trains the model** for `20 epochs`, with `batch_size=32`.
- Uses `validation_data=(X_test, y_test)` to **monitor validation accuracy/loss**.

✅ If you **want fast and simple embeddings** → Use **Word2Vec/GloVe**.

✅ If you **want best accuracy & context-awareness** → Use **Raw BERT Word Embeddings**.

```
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.3979 - loss: 1.0696 - val_accuracy: 0.3750 - val_loss: 1.0995
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.4404 - loss: 1.0565 - val_accuracy: 0.3750 - val_loss: 1.1092
<keras.src.callbacks.history.History at 0x7b95eb856050>
```

training accuracy is 44%, and validation accuracy is very low (37.50%),

**validation loss is very high (1.1092).

Increasing a Epoch Training Accuracy Increase but Validation Accuracy Reduced

⚒️ How to Fix This?

Here are some key steps to improve generalization and validation performance:

1️⃣ Reduce Overfitting

✅ **Increase Dropout Rate:**

Your dropout is currently **0.3**. Try increasing it to **0.5** to regularize learning.

✅ **Use L2 Regularization:**

Add `kernel_regularizer=tf.keras.regularizers.l2(0.01)` to your LSTM layers.

✅ **Reduce Model Complexity:**

Your LSTM model might be too complex for the dataset. Try reducing the number of LSTM units.

2️⃣ Tune Hyperparameters

✅ Reduce Learning Rate

The Adam optimizer's default learning rate is **0.001**. Try lowering it:

python Copy code

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
```

✅ Increase Training Data (Augmentation)

If possible, increase the dataset size with **data augmentation**.

✅ Early Stopping:

Stop training when the validation loss stops improving.

python Copy code

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, res
```

Another Try to Prevent Overfit Use L2 - Ridge Regularization

```
# Define LSTM Model with Fixes
model = Sequential([
    LSTM(64, return_sequences=True, kernel_regularizer=l2(0.01), input_shape=(X.shape[1], X.shape[2])),
    Dropout(0.5),
    LSTM(32, kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(3, activation='softmax')
])

# Compile with lower learning rate
optimizer = Adam(learning_rate=0.0005)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train with early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test),
callbacks=[early_stopping])
```

This version **improves the previous LSTM model** with **regularization, better optimization, and early stopping**.

🔹 Key Enhancements Compared to Previous Model:

1️⃣ **L2 Regularization (`kernel_regularizer=l2(0.01)`)** → Reduces overfitting by penalizing large weights.

2️⃣ **Increased Dropout (`0.5` instead of `0.3`)** → Further prevents overfitting.

3️⃣ **Smaller LSTM Layers (`64 → 32` units)** → Reduces complexity while maintaining performance.

4️⃣ **Lower Learning Rate (`0.0005` instead of default `0.001`)** → Stabilizes training and improves generalization.

5️⃣ **Early Stopping (`patience=5`)** → Stops training when validation loss stops improving, preventing unnecessary epochs.

🚀 **Overall Impact: Better regularization, optim… training, and improved generalization!**

```
Epoch 50/50
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.4188 - loss: 1.0766 - val_accuracy:
0.3900 - val_loss: 1.0870

<keras.src.callbacks.history.History at 0x7b95f4d3add0>
```

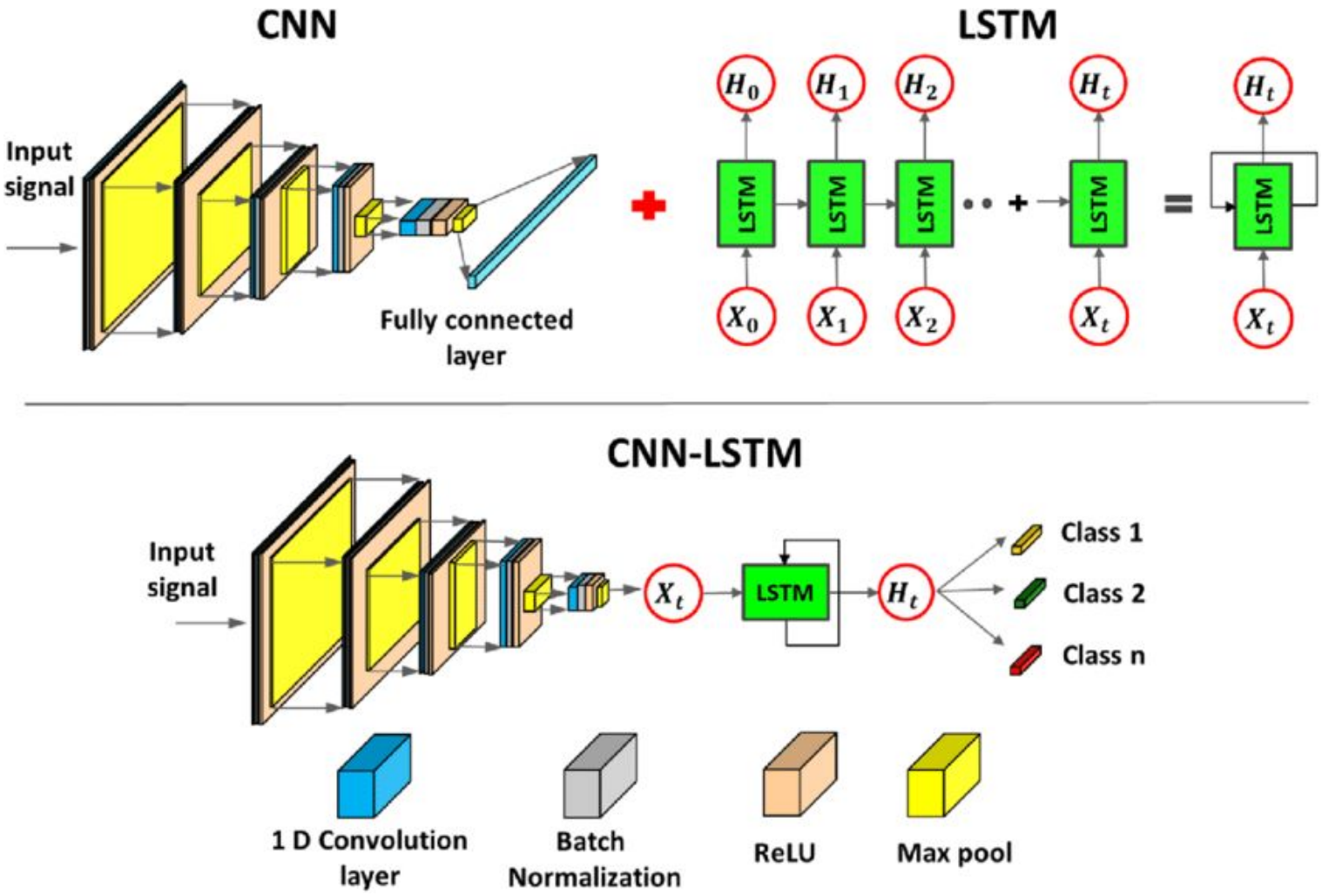
Still Low Accuracy

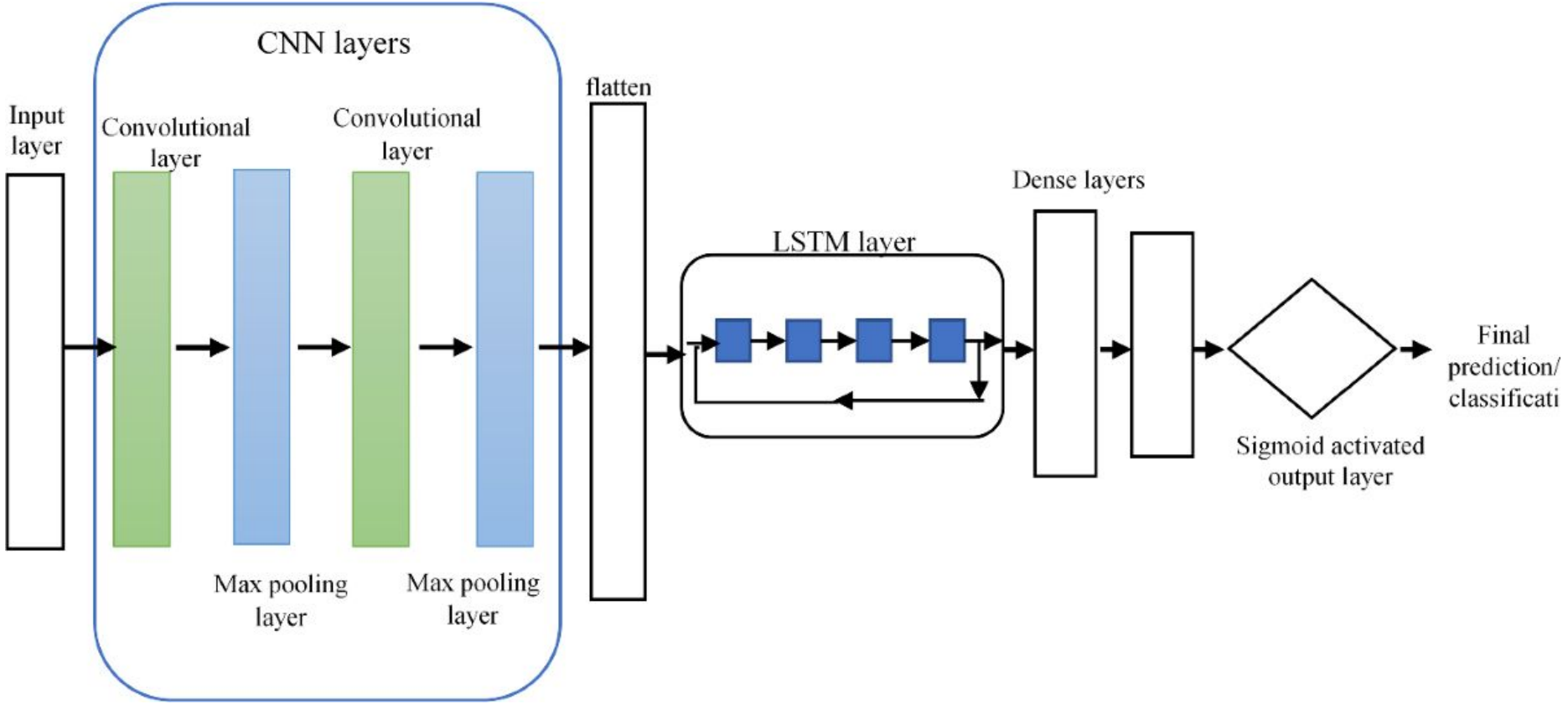
2️⃣ Try a CNN-LSTM Hybrid

LSTMs are great for sequential dependencies, but **text embeddings don't have clear sequential structures**. Adding a **CNN layer** can help extract features before passing them to LSTM.

- **New Model: CNN + LSTM**

CNN-LSTM hybrid architecture because it combines Conv1D (CNN) and LSTM layers in a sequential manner.


CNN
LSTM
Input signal
Fully connected layer
H_0
H_1
H_2
H_t
H_t
LSTM
X_0
X_1
X_2
X_t
X_t
CNN-LSTM
Input signal
X_t
LSTM
H_t
Class 1
Class 2
Class n
1 D Convolution layer
Batch Normalization
ReLU
Max pool


CNN layers
Input layer
Convolutional layer
Convolutional layer
Max pooling layer
Max pooling layer
flatten
LSTM layer
Dense layers
Sigmoid activated output layer
Final prediction/ classificati

```
# Define the Model
model = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(768, 1)),
    MaxPooling1D(pool_size=2),
    LSTM(64, return_sequences=True),
    Dropout(0.5),
    LSTM(32),
    Dropout(0.5),
    Dense(3, activation='softmax')
])

# Compile Model
model.compile(optimizer=Adam(learning_rate=0.0005), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

This code **combines CNN (Conv1D) with LSTM** for sentiment classification using **BERT embeddings**.

🔹 Key Improvements & Steps:

1️⃣ **Load Data** → Reads `df2_bert.csv`, converts BERT embeddings from string to NumPy arrays.

2️⃣ **Reshape Data** → Adjusts shape for `Conv1D` to `(num_samples, 768, 1)`, treating **each embedding as a time series**.

3️⃣ **Train-Test Split** → Splits dataset (80% train, 20% test).

4️⃣ **Model Architecture:**

- **Conv1D (64 filters, kernel=3)** → Extracts local features.
- **MaxPooling1D** → Downsamples feature maps.
- **LSTM (64 → 32 units)** → Captures sequential patterns.
- **Dropout (0.5)** → Prevents overfitting.
- **Dense(3, softmax)** → Classifies into **Negative, Neutral, Positive**.

5️⃣ **Compile Model** → Uses **Adam (lr=0.0005)** for stable learning.

6️⃣ **Early Stopping** → Stops training if `val_loss` doesn't improve for **5 epochs**.

7️⃣ **Train Model** → Runs for **up to 50 epochs** with batch size **32**.

🚀 **Impact:** CNN extracts local features, LSTM captures long-term dependencies → **Better sentiment classification!**

A **3-sized kernel** means the filter looks at **3 consecutive values** at a time.

📌 Types of Pooling

1️⃣ Max Pooling (`MaxPooling1D` or `MaxPooling2D`)

- Takes the **maximum value** from a region of the feature map.
- Helps keep the **strongest** features while discarding less useful ones.

Example (`pool_size=2`):

Input: `[2, 5, 8, 3, 7, 1]`

After **Max Pooling (size=2, stride=2):**

👉 `[5, 8, 7]` (Takes the **maximum** of each pair: `[2,5] → 5`, `[8,3] → 8`, `[7,1] → 7`)

2️⃣ Average Pooling (`AveragePooling1D`)

- Takes the **average** of values in a region.
- Smoother than max pooling but **less effective** for detecting strong signals.

Example (`pool_size=2`):

Input: `[2, 5, 8, 3, 7, 1]`

After **Average Pooling (size=2, stride=2):**

👉 `[(2+5)/2, (8+3)/2, (7+1)/2] = [3.5, 5.5, 4]`

Conclusion of 2nd Try

Testing so Much Type like Machine Learning Algorithm and Using Deep learning

Like FNN,RNN, LSTM, LSTM + CNN Hybrid and Using L2 Regularization to Prevent Overfit and Try to Change Dence and Increase a Learning Rate Like adam and Epoch Increase to number of Cycles and Useing Dropout to prevent Overfit - after so much accuracy not increase

Finally i Realize my Mistake Because I am Using Synthetic 1000 Rows Datasets so its My Mistakes

Using Real Time Dataset - Financial Analysis

5000 Rows Dataset - Get Accuracy 61% Using of All Steps of Previous One

And Prediction Little But Struggle

Preprocessing / Feature Selection / Model Training - 3

Using Real Time Dataset - **Twitter Sentiment From Kaggle**

Preprocessing

```
[ ]
1 import kagglehub
2
3 # Download latest version
4 path = kagglehub.dataset_download("jp797498e/twitter-entity-sentiment-analysis")
5
6 print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/jp797498e/twitter-entity-sentiment-analysis?dataset_version_number=2...
100%|██████████| 1.99M/1.99M [00:00<00:00, 94.4MB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/jp797498e/twitter-entity-sentiment-analysis/versions/2
```

```
[ ]
1 import pandas as pd
```

```
[ ]
1 df2=pd.read_csv('/content/twitter_training.csv')
```

```
[ ]
# Define column names
column_names = ["column_1", "column_2","Sentiment","Sentence"]  # Replace with actual column names

# Load the CSV file without header and assign column names
df2 = pd.read_csv("/content/twitter_training.csv", header=None, names=column_names)

print(df2.head())  # Check if columns are correct
```

```
   column_1     column_2 Sentiment  \
0      2401  Borderlands  Positive
1      2401  Borderlands  Positive
2      2401  Borderlands  Positive
3      2401  Borderlands  Positive
4      2401  Borderlands  Positive

                                            Sentence
0  im getting on borderlands and i will murder yo...
1  I am coming to the borders and I will kill you...
2  im getting on borderlands and i will kill you ...
3  im coming on borderlands and i will murder you...
4  im getting on borderlands 2 and i will murder ...
```

```
[ ]  1 df2.shape
```

```
(74682, 4)
```

Missing Values

```
  1 df2.isnull().sum()
```

	0
column_1	0
column_2	0
Sentiment	0
Sentence	686

dtype: int64

```
df2['Sentiment'].value_counts()
```

Sentiment	count
Negative	22542
Positive	20832
Neutral	18318
Irrelevant	12990

dtype: int64

```
df2 = df2[df2['Sentiment'] != 'Irrelevant']
```

```
print(df2['Sentiment'].value_counts())  # Should no longer show "Irrelevant"
```

```
Sentiment
Negative    22542
Positive    20832
Neutral     18318
Name: count, dtype: int64
```

```
df2.isnull().sum()
```

	0
column_1	0
column_2	0
Sentiment	0
Sentence	571

dtype: int64

```
df2 = df2.dropna(subset=['Sentence'])
```

```
print(df2.isnull().sum())  # Check for remaining missing values
```

```
column_1     0
column_2     0
Sentiment    0
Sentence     0
```

```
[ ]  1 df2.shape
```

```
(61121, 4)
```

```
[ ]  1 df2_clear=pd.read_csv('/content/df2_clear_twitter.csv')
```

```
  1 df2_clear.head()
```

	Unnamed: 0	column_1	column_2	Sentiment	Sentence
0	0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...
1	1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...
2	2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...
3	3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...
4	4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...

Text Classification

	Unnamed: 0	column_1	column_2	Sentiment	Sentence	Cleaned_Review
0	0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...	im getting borderlands murder
1	1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...	coming borders kill
2	2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...	im getting borderlands kill
3	3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...	im coming borderlands murder
4	4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im getting borderlands murder

✅ **Removes Stopwords**

✅ **Tokenizes & Converts to Lowercase**

✅ **Applies POS Tagging with NLTK**

✅ **Keeps Only Important Words**

✅ **Returns the Processed Text**

1) Feature Extraction

Vectorization - BERT

Am Using BERT Embedding instead of Word2Vec or TF-IDF

Term Frequency-Inverse Document Frequency

Feature	BERT 🧠	Word2Vec 🔄	TF-IDF 📊
Type	Contextual Embeddings	Static Embeddings	Statistical Method
Understands Context?	✅ Yes (Considers word meaning based on surrounding words)	❌ No (Same word = Same vector)	❌ No (Just word frequency)
Handles Polysemy (same word, different meanings)?	✅ Yes	❌ No	❌ No
Embeddings	Dynamic (Different for each sentence)	Fixed (Same for every context)	N/A (No embeddings, just word importance scores)
Training Approach	Transformer-based (Pretrained on massive text data)	Neural Network (Trained on word co-occurrence)	Simple formula-based (TF & IDF scores)
Output Dimension	Large (768 for BERT-base)	Smaller (100-300 typically)	Sparse matrix (High-dimensional)
Best for?	NLP tasks requiring deep language understanding (Chatbots, Sentiment Analysis, Q&A)	Word similarity, basic NLP tasks	Text Search, Keyword Extraction

- **Use BERT** 🧠 if you need deep language understanding (e.g., sentiment analysis, text classification).
- **Use Word2Vec** 🔄 if you need **fast word similarity** (e.g., recommendation systems, synonyms).
- **Use TF-IDF** 📊 if you just need a **simple keyword-based approach** (e.g., search engines, topic modeling).

🚀 **Summary:**

BERT is the most advanced, but **Word2Vec is lightweight**, and **TF-IDF is the simplest**!

```python
from transformers import BertTokenizer, BertModel
import torch  # Required for tensor operations

# 3. BERT Embeddings
bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embedding(text):
    inputs = bert_tokenizer(text, return_tensors='pt', padding=True, truncation=True)
    outputs = bert_model(**inputs)
    last_hidden_state = outputs.last_hidden_state
    return list(last_hidden_state.mean(dim=1).detach().numpy()[0])

df2_pre['BERT_Embedding'] = df2_pre['Cleaned_Review'].apply(get_bert_embedding)
```

1. **Loads BERT tokenizer & model** (`bert-base-uncased`).
2. **Tokenizes text** (`return_tensors='pt'` for PyTorch, padding, truncation).
3. **Passes tokenized text through BERT** to get embeddings.
4. **Extracts** `last_hidden_state` (word-level embeddings).
5. **Computes sentence-level embedding** by **averaging word embeddings** (`mean(dim=1)`).
6. **Stores embeddings in** `df2_pre['BERT_Embedding']` for each cleaned review.

Sentiment	Sentence	Cleaned_Review	BERT_Embedding
Positive	im getting on borderlands and i will murder yo...	im getting borderlands murder	[0.20229957, -0.11369792, 0.39551067, -0.01954...
Positive	I am coming to the borders and I will kill you...	coming borders kill	[0.3408045, -0.08364203, -0.020582985, 0.09118...
Positive	im getting on borderlands and i will kill you ...	im getting borderlands kill	[0.12905084, -0.1902146, 0.3261264, -0.0198617...
Positive	im coming on borderlands and i will murder you...	im coming borderlands murder	[0.11887888, -0.010340976, 0.5494281, -0.09218...
Positive	im getting on borderlands 2 and i will murder ...	im getting borderlands murder	[0.20229957, -0.11369792, 0.39551067, -0.01954...

```
1 import numpy as np
2 embeddings_array = np.stack(df2_bert['BERT_Embedding'].values)
3 print(embeddings_array.shape)  # (num_samples, embedding_dim)
4
```

```
(59585, 768)
```

BERT have 768 Dimension

1️⃣ **Converts BERT embeddings** (lists) into a DataFrame.

2️⃣ **Applies Truncated SVD** (`n_components=50`) to reduce embeddings from high-dimensional space.

3️⃣ **Replaces original embeddings** with the reduced version (50D).

4️⃣ **Saves the SVD model** (`twitter_real_svd_model.pkl`) for future use.

🚀 **Purpose: Dimensionality reduction** to make embeddings more efficient for modeling!

```
]    1 import numpy as np
     2 embeddings_array = np.stack(df2_bert['BERT_Embedding'].values)
     3 print(embeddings_array.shape)  # (num_samples, embedding_dim)
```

```
(59585, 50)
```

```
[ ]    1 print(df2_bert['BERT_Embedding'].dtype)  # Check data type
       2 print(df2_bert['BERT_Embedding'].iloc[0])  # Inspect the first row
```

```
object
[-5.5548859e+00 -1.8198940e+00  5.0649059e-01 -7.8539181e-01
  4.6617353e-01 -3.0169958e-01 -4.2895412e-01 -2.3148698e-01
  4.2453995e-01  3.8952893e-01  5.4605657e-01 -2.9523316e-01
 -3.0411294e-01  3.0860978e-01 -6.4861113e-01 -5.7338023e-01
  1.0317008e+00 -5.8545899e-01  4.5708302e-01 -3.0512393e-01
  5.0695896e-01 -5.2303708e-01 -5.7385457e-01 -7.9525393e-01
 -3.9934093e-01  2.2154985e-01 -6.1850101e-01  5.7615167e-01
 -1.4323318e-01  2.2578627e-02 -3.8158482e-01  7.8426492e-01
  3.3592805e-03  2.0836334e-01 -4.4404042e-01  2.0759439e-01
 -3.5259080e-01  1.7114232e-01 -1.9267359e-01  4.3996865e-01
  2.8727388e-01 -4.4300443e-01  5.6378925e-01  3.2159045e-02
 -2.8057790e-01  2.2536144e-01 -4.5820083e-03  4.2081207e-01
  6.7168377e-02  3.2753371e-02]
```

```
[ ]    1 feature = df2_bert['BERT_Embedding']
       2 target = df2_bert['Sentiment']
```

Model Training

1) Random Forest Algorithm

```
Train Accuracy: 0.99
Test Accuracy: 0.75

Classification Report (Test Set):
              precision    recall  f1-score   support

           0       0.74      0.83      0.78      4424
           1       0.77      0.62      0.69      3497
           2       0.74      0.77      0.75      3996

    accuracy                           0.75     11917
   macro avg       0.75      0.74      0.74     11917
weighted avg       0.75      0.75      0.74     11917

🚨 Model is OVERFITTING: High training accuracy, lower test accuracy.
```

F1-score score based on how Precision and Recall Score was balanced or not , recall is more important than recall because of recall is calculated by Actual Positive value (TP) and Predicted Negative (FN) , Recall by Actual (TP)and Predicted Positive (FP), Maintain TP higher than FP because of Reduce False Alarm - using ROC Cureve Area, - Macro AVG is Calculated by based on Classes majority or Minority but Weighted AVG is Calculated by Balanced of Each Classes

✅ **F1-score measures the balance between Precision and Recall.**

✅ **Recall is often more important than Precision in cases where False Negatives (FN) are costly.**

✅ **Precision = TP / (TP + FP) → Focuses on reducing False Positives (FP).**

✅ **Recall = TP / (TP + FN) → Focuses on reducing False Negatives (FN).**

✅ **Reducing FP helps lower false alarms, and the ROC Curve (AUC-ROC) helps assess model performance across different thresholds.**

✅ **Macro Average F1-score treats all classes equally (good for imbalanced data analysis).**

✅ **Weighted Average F1-score accounts for class imbalance by weighting each class's F1-score based on its sample count.**

🚀 **Summary:** If you want to **reduce false alarms (FP)**, focus on **Precision**. If you want to **minimize missing positive cases (FN)**, focus on **Recall**. The choice depends on the problem!

Files

Analyze your files with code written by Gemini Upload

..
.ipynb_checkpoints
sample_data
df2_clear.csv
df2_clear_twitter.csv
df2_twitter_preprocessed.csv
twitter_bert_embedding.csv
twitter_real_svd_model.pkl
twitter_training.csv

						window partition is ...	years nvidia dri...	-0.087...
61118	61118	74679	9200	Nvidia	Positive	Just realized the windows partition of my Mac ...	realized windows partition mac years nvidia dr...	[-0.387785, -0.12202826, 0.49094146, -0.104739...
						Just realized		

Next steps: Generate code with df2_pre | View recommended plots | New interactive sheet

```
[ ]  1 df2_pre.to_csv('twitter_bert_embedding.csv')
```

```
[4]  1 df2_bert=pd.read_csv('/content/twitter_bert_embedding.csv')
```

17s

```
1 df2_bert.head()
```

Unnamed: 0.2	Unnamed: 0.1	Unnamed: 0	column_1	column_2	Sentiment	Sentence	Cleaned_Review	BERT_Embed

Am Trying XGBoosting, Naive bayes, FNN Accuracy no Increasing - High Ram Issues so I am Trying LSTM & Extreme CNN + LSTM Hybrid

```python
import joblib

# Save the trained model
joblib.dump(rf_classifier, "random_forest_model.pkl")
print("Model saved successfully!")

```

```
Model saved successfully!
```

Future Prediction

```
[ ]  1 import joblib
```

```
[ ]  1 # Load the saved model
     2 rf_classifier_loaded = joblib.load("random_forest_model.pkl")
     3 print("Model loaded successfully!")
```

```
Model loaded successfully!
```

```
 1 import joblib
 2 import pandas as pd
 3 import numpy as np
 4 from sentence_transformers import SentenceTransformer
 5
 6 # Load the trained RandomForest model
 7 rf_classifier_loaded = joblib.load("random_forest_model.pkl")
 8 print("Model loaded successfully!")
 9
10 # Load the same BERT model used for embedding
11 bert_model = SentenceTransformer('bert-base-uncased')
12 print("BERT model loaded successfully!")
13
14 # Load the SVD model if dimensionality reduction was used
15 svd = joblib.load("/content/twitter_real_svd50D_model.pkl")  # Skip this if SVD wasn't used
16 print("SVD model loaded successfully!")
17
```

```python
# Function to get BERT embedding
def get_bert_embedding(sentence):
    embedding = bert_model.encode(sentence)
    return embedding

# Example user input
user_sentence = "This insurance policy is very helpful."

# Convert input sentence to BERT embedding
bert_embedding = get_bert_embedding(user_sentence)

# Convert embedding to DataFrame
bert_embedding_df = pd.DataFrame([bert_embedding])

# Apply SVD transformation (skip if not used in training)
bert_embedding_reduced = svd.transform(bert_embedding_df)  # Should reduce to 50D if used

# Create DataFrame for model prediction
new_data = pd.DataFrame(bert_embedding_reduced)

# Make a prediction
predicted_class = rf_classifier_loaded.predict(new_data)[0]

# Class mapping (adjust based on your dataset)
class_labels = {0: "Negative review", 1: "Neutral review", 2: "Positive review"}

```

```
config.json: 100% 570/570 [00:00<00:00, 41.5kB/s]
model.safetensors: 100% 440M/440M [00:07<00:00, 72.2MB/s]
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 4.35kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 1.43MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.85MB/s]
BERT model loaded successfully!
SVD model loaded successfully!

Predicted class: 2 (Positive review)
```

Using Real Time Review About OPPO Mobile From Amazon

```
# Example user input
user_sentence = """This is such a worst phone i.got.... it is countinouly hanging it's only been 1 month and the problems started already
 ... My hard income I spend here is gone..... Totally waste and value less product.... My trust for flipkart is getting low every single day""
```

```
BERT model loaded successfully!
SVD model loaded successfully!

Predicted class: 0 (Negative review)
```

Perfectly Predicted as Negative

Prediction Successful

My Experience is Synthetic dataset very tough to get accuracy and prediction - this nlp project need real time with large number of dataset so machine will learn easily on real time dataset and machine learning not get accuracy means deep learning not get enough accuracy its based on 10 points only difference and this project time and RAM / Internet Consuming very High - Computationally Expensive - BERT is Best and Dimension reduce to 50D is very Good all of the Information was Captured

Conclusion:

While working on this project, I gained valuable experience in integrating NLP techniques with machine learning. I learned how to use **BERT** for generating sentence embeddings, apply **Truncated SVD** for dimensionality reduction, and build a **Random Forest classifier** for sentiment classification. Additionally, I improved my skills in **data preprocessing, feature engineering, and model deployment** using Python libraries like **pandas, joblib, and scikit-learn**. This project also enhanced my understanding of how to handle real-world text data and optimize models for better predictions.

Thank You