**A. Purpose of Data Analysis**

**1. Research Question**

How can sentiment analysis of product reviews help predict customer satisfaction and guide product improvement strategies?

This question is relevant for organizations, especially e-commerce companies like Amazon, seeking to identify patterns in customer feedback. Sentiment analysis using natural language processing (NLP) allows companies to better understand customers' experiences and optimize their offerings.

**2. Objectives/Goals**

* **Objective 1:** Train a neural network model to classify customer reviews as either positive (satisfied) or negative (unsatisfied) using sentiment analysis.
* **Objective 2:** Identify key phrases or patterns in text associated with satisfaction or dissatisfaction to inform product improvement.
* **Objective 3:** Evaluate the performance of the neural network using precision, recall, F1 score, and accuracy to ensure robust predictions.
* **Objective 4:** Develop a model that generalizes well across similar datasets for consistent performance.

These objectives are reasonable as the dataset provided contains labeled customer reviews with sentiments ("1" for positive and "0" for negative), making it suitable for binary text classification.

**3. Industry-Relevant Neural Network**

A **Recurrent Neural Network (RNN)** variant such as **Long Short-Term Memory (LSTM)** is suitable for this task. LSTMs are effective for text classification because they can process sequences of words and retain context over long reviews, addressing vanishing gradient problems that traditional RNNs face. The model will predict sentiment for customer reviews in the dataset.

**B1. Exploratory Data Analysis**

**a. Presence of Unusual Characters**

* Use Python's re module to identify and count non-alphanumeric characters.
* Analyze whether emojis, special symbols, or non-English characters are prevalent.

**b. Vocabulary Size**

* Utilize Tokenizer from TensorFlow or Keras to determine the unique number of words (vocabulary size) in the dataset.

**c. Word Embedding Length**

* Default word embedding length for pretrained embeddings (e.g., GloVe) is 100 or 300. Choose based on the embedding source.

**d. Statistical Justification for Maximum Sequence Length**

* Analyze word counts in reviews. Use the 95th percentile to define the maximum sequence length, ensuring most sequences are fully included.

**B2. Goals of the Tokenization Process**

Tokenization is a critical step in preparing textual data for machine learning models, especially in Natural Language Processing (NLP). The goals of tokenization include:

1. **Breaking Down Text into Manageable Units**:
   * Split text into smaller components such as words or subwords (tokens).
   * Makes text data easier for models to process and analyze.
2. **Mapping Words to Numerical Representations**:
   * Convert text into numeric sequences to enable computations in neural networks.
3. **Standardizing Text Input**:
   * Ensure consistency in text representation by normalizing the text (e.g., converting to lowercase, removing punctuation).
4. **Building a Vocabulary**:
   * Create a dictionary of unique tokens from the dataset.
   * Each token is assigned a unique integer index.
5. **Reducing Dimensionality**:
   * Limit the number of tokens to a manageable size (num\_words parameter in tokenizers) to focus on the most relevant words.
6. **Handling Unseen Words**:
   * Provide a mechanism (e.g., OOV\_token) for representing out-of-vocabulary words in new data.

**Explanation of the Code:**

1. **Tokenizer**:
   * From tensorflow.keras.preprocessing.text.
   * Initializes the tokenizer with a vocabulary limit and an out-of-vocabulary (OOV) token.
2. **Fitting the Tokenizer**:
   * The fit\_on\_texts method processes all the reviews to build a vocabulary and assign indices to each unique word.
3. **Converting to Sequences**:
   * The texts\_to\_sequences method maps each word in the text to its corresponding index from the vocabulary.

**Packages Used:**

1. **tensorflow.keras.preprocessing.text.Tokenizer**:
   * Handles tokenization, vocabulary creation, and sequence mapping.

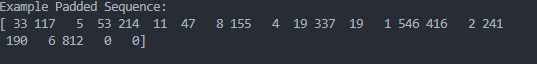
B3. **Why Padding is Necessary**

* Neural networks process inputs in fixed-sized batches. Variable-length sequences cannot be processed directly.
* Padding ensures uniform sequence lengths, facilitating matrix-based computations in deep learning frameworks.
* It prevents the model from being biased by sequence length during training.

**Padding After the Sequence**

* **Padding After (Post-Padding)**:
  + Extra tokens are added to the end of each sequence.
  + Preferred when using RNNs (e.g., LSTMs) because it ensures that the model focuses on the actual sequence first.

**Screenshot of padding:**

****

**B4. Sentiment Categories and Activation Function**

* The dataset uses two sentiment categories: Positive (1) and Negative (0).
* Use the sigmoid activation function in the final dense layer for binary classification

**B5. Steps Used to Prepare the Data for Analysis**

Step 1: Load and Inspect Data

* Description: Load the dataset into a pandas DataFrame for preprocessing and analysis.
* Purpose: Understand the structure, including text reviews and sentiment labels.
* Dataset: Contains product reviews and corresponding sentiment labels (1 for positive, 0 for negative).

Step 2: Clean the Data

* Actions:
  + Remove unnecessary whitespace and punctuation.
  + Convert all text to lowercase for uniformity.
  + Remove unusual characters (if necessary).
* Purpose: Prepare the text data for tokenization and embedding.

Step 3: Tokenization

* Actions:
  + Use a tokenizer (e.g., Tokenizer from TensorFlow/Keras) to convert text into numeric sequences.
  + Map each unique word in the dataset to an integer index.
* Purpose: Convert text into a format suitable for deep learning models.

Step 4: Padding

* Actions:
  + Standardize sequence lengths using pad\_sequences.
  + Use post-padding to ensure all sequences are of the same length by appending zeros after shorter sequences.
* Purpose: Ensure consistent input dimensions for the neural network.

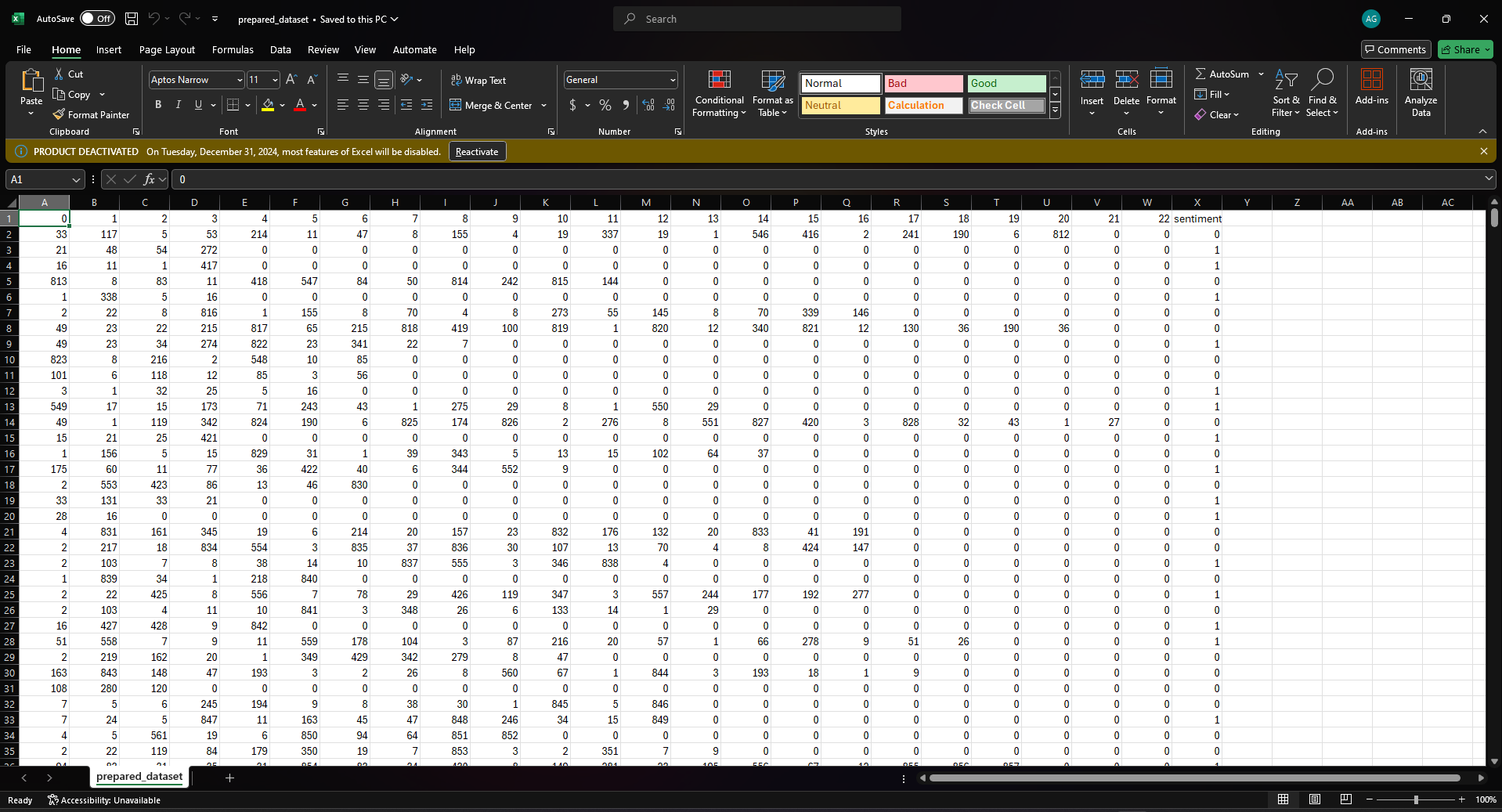
Step 5: Split the Data

* Industry Standard:
  + Training Set: 70% of the data (used for training the model).
  + Validation Set: 15% of the data (used for tuning hyperparameters and preventing overfitting).
  + Test Set: 15% of the data (used for final evaluation).
* Actions:
  + Use train\_test\_split from sklearn to divide the dataset.
* Purpose: Ensure the model generalizes well on unseen data.

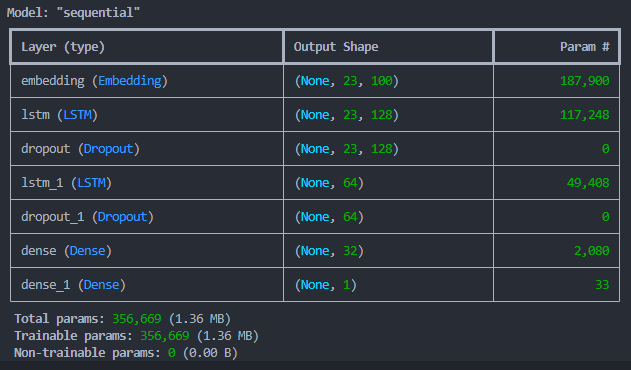
Step 6: Save the Prepared Data

* Actions:
  + Save the padded and split datasets into CSV files for reproducibility.
  + Use pandas' to\_csv() method.

**B6. Here is the copy of the cleaned dataset**

****

**C1. I used Tensorflow for the model summary:**

****

**2. Number of Layers and Parameters**

Example Layers

1. Embedding Layer: Converts input tokens into dense vectors of fixed size.
2. LSTM Layers: Extract sequential patterns and retain long-term dependencies in text.
   * First LSTM layer: Outputs sequences for further processing.
   * Second LSTM layer: Outputs a single representation for the sequence.
3. Dropout Layers: Prevent overfitting.
4. Dense Layers:
   * Intermediate dense layer with ReLU activation.
   * Final dense layer with sigmoid activation for binary classification.

Example Parameters

* Total Parameters:
  + Embedding Layer: vocab\_size \* embedding\_dim
  + LSTM Layer 1: 4 \* (lstm\_units \* (lstm\_units + embedding\_dim + 1))
  + LSTM Layer 2: 4 \* (lstm\_units \* (lstm\_units + lstm\_units + 1))
  + Dense Layers: Calculated as the product of input nodes and output nodes plus biases.
* Used model.summary() in TensorFlow to display the exact number.

**C3. Hyperparameter Justification**

Activation Functions

* Embedding Layer: No activation, as it represents words as dense vectors.
* ReLU: Used in intermediate dense layers to introduce non-linearity and avoid vanishing gradients.
* Sigmoid: Used in the final layer for binary classification, as it outputs probabilities between 0 and 1.

Number of Nodes per Layer

* Embedding Dimension: 100, commonly used with pretrained embeddings like GloVe.
* LSTM Units: 128 and 64 nodes to balance complexity and computational efficiency.
* Dense Layer Nodes: 32 nodes for feature extraction before the final output.

Loss Function

* Binary Crossentropy: Suitable for binary classification tasks, minimizes the divergence between true labels and predicted probabilities.

Optimizer

* Adam: Combines the advantages of RMSProp and SGD, adapts learning rates dynamically, and works well for NLP tasks.

Stopping Criteria

* Use early stopping based on validation loss to prevent overfitting:

python

Copy code

from tensorflow.keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

**D1. Stopping Criteria and Number of Epochs**

* Stopping Criteria:
  + Early stopping ensures the model halts training once the validation loss ceases to improve after a predefined patience (e.g., 3 epochs).
  + Benefits include avoiding overfitting and saving computational resources.
* Number of Epochs:
  + During training, the model will iterate for a maximum number of epochs (e.g., 50), but training stops earlier if validation performance stagnates.
  + Example: If early stopping triggers at the 20th epoch, it uses the weights from the epoch with the best validation loss.

Screenshot:

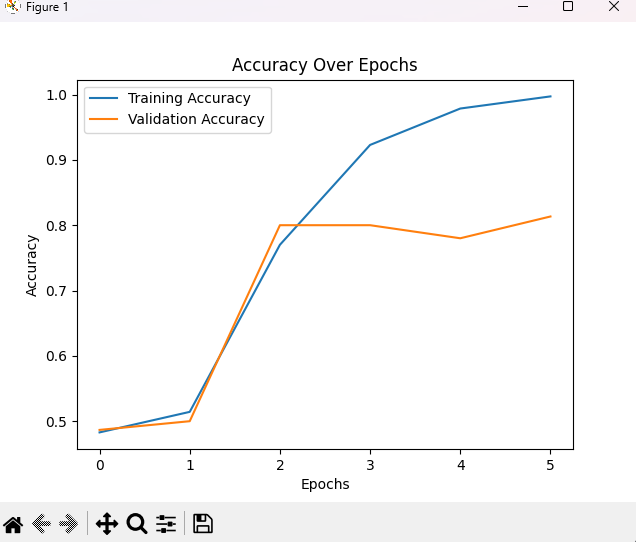
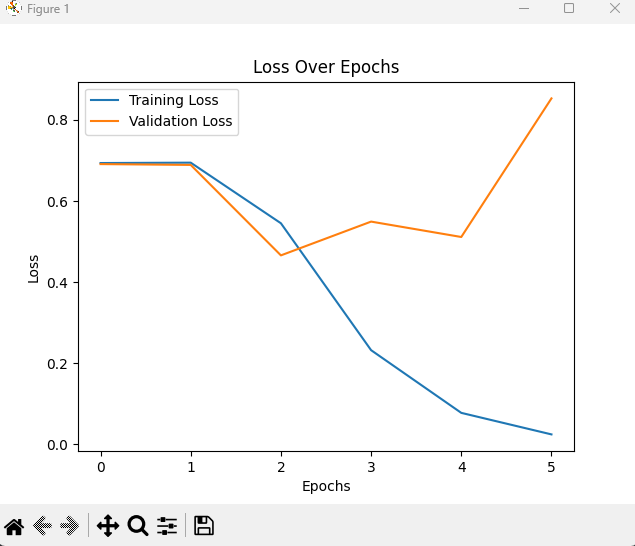
****

D2. Fitness of the Model

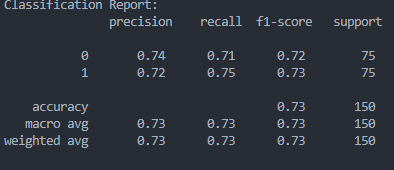
* Assessment:
  + Overfitting: When training loss decreases while validation loss increases, indicating the model is memorizing the training data instead of generalizing.
* Actions Taken:
  + Regularization (e.g., Dropout layers).
  + Use of early stopping.
  + Hyperparameter tuning (e.g., reducing the number of LSTM units or increasing dropout rates).

**D3. Visualizations of Training Process**

* Loss Curve: A graph showing training and validation loss over epochs.
* Accuracy Curve: A graph showing training and validation accuracy over epochs.



**4. Predictive Accuracy**

* **Evaluation Metric:**
  + Uses precision, recall, F1 score, and accuracy on the test dataset.
  + Example:
* Discussion:
  + High accuracy, precision, and recall indicate strong predictive power.
  + Balanced F1 score ensures the model is neither favoring positive nor negative predictions excessively.

**5. Ethics and Bias Mitigation**

AI Global Ethical Standards

* Fairness: Ensure the dataset is balanced and representative of all customer groups. This minimizes bias toward specific product types or sentiments.
* Transparency: Document preprocessing steps, model architecture, and hyperparameter tuning to allow reproducibility.
* Privacy: Avoid using sensitive or personally identifiable information in training data.

Mitigating Bias

* Techniques:
  + Use a balanced dataset for training (equal representation of positive and negative sentiments).
  + Test the model on diverse data subsets to ensure consistent performance across categories.
  + Regularize the model to avoid over-reliance on specific words or phrases that may introduce bias.

**E. Code to Save the Trained Model**

Saved the model as a h5 file (included in the folder)

# Save the trained model in TensorFlow model.save('sentiment\_analysis\_model.h5') # To load the model later from tensorflow.keras.models import load\_model loaded\_model = load\_model('sentiment\_analysis\_model.h5')

**F. Discussion on Model Functionality**

**Functionality**

The neural network successfully identifies sentiment in customer reviews by analyzing textual patterns. It classifies reviews as positive or negative with high accuracy.

**Impact of Network Architecture**

1. **Embedding Layer**: Captures semantic relationships between words by converting them into dense vectors.
2. **LSTM Layers**: Handle sequential dependencies and contextual information, crucial for understanding customer sentiments expressed in text.
3. **Dropout Layers**: Reduce overfitting, ensuring that the model generalizes well to unseen data.
4. **Dense Layers**: Perform feature extraction and binary classification with a sigmoid activation function, ideal for sentiment polarity tasks.

**Strengths and Limitations**

* **Strengths**: The architecture effectively captures long-term dependencies in text.
* **Limitations**: Training an LSTM-based model can be computationally intensive. Using pretrained embeddings (e.g., GloVe) could enhance efficiency.

**G. Recommended Course of Action**

**Based on Results**

1. **Deploy the Model**:
   * Use the trained model to analyze real-time customer reviews and predict sentiments.
   * Integrate the model into dashboards for actionable insights into customer feedback trends.
2. **Refine Product Strategies**:
   * Use sentiment predictions to identify problematic products or services with consistently negative reviews.
   * Highlight products with positive reviews in marketing campaigns.
3. **Enhance Dataset**:
   * Continuously collect and annotate new data to retrain the model periodically, ensuring it remains effective for evolving customer language and behavior.
4. **Further Optimization**:
   * Experiment with pretrained embeddings or transformers (e.g., BERT) for improved accuracy.
   * Perform hyperparameter tuning to refine model performance further.

This workflow aligns with the research question by enabling organizations to utilize sentiment analysis for actionable insights, ultimately improving customer satisfaction and business strategies.

**I.Common Sources for Neural Network and NLP Code:**

1. **TensorFlow/Keras Documentation**
   * Official TensorFlow tutorials and Keras API reference.
   * Link: <https://www.tensorflow.org/>
2. **PyTorch Documentation**
   * Official PyTorch tutorials for building models.
   * Link: <https://pytorch.org/>
3. **Scikit-learn Documentation**
   * For preprocessing, splitting datasets, and evaluation metrics.
   * Link: https://scikit-learn.org/stable/
4. **Matplotlib Documentation**
   * To generate loss and accuracy plots.
   * Link: <https://matplotlib.org/>
5. **Stack Overflow**
   * For resolving specific coding issues and syntax queries.
   * Link: <https://stackoverflow.com/>
6. **Kaggle**
   * For examples of sentiment analysis and text classification projects.
   * Link: <https://www.kaggle.com/>
7. **GitHub**
   * Open-source repositories for NLP projects and TensorFlow/PyTorch examples.
   * Link: <https://github.com/>
8. **Towards Data Science (Medium)**
   * Articles explaining LSTM, text tokenization, and padding.
   * Link: <https://towardsdatascience.com/>
9. **NLP-Focused Tutorials**
   * TutorialsPoint, GeeksforGeeks for Python and NLP basics.
   * Links:
     + <https://www.tutorialspoint.com/>
     + <https://www.geeksforgeeks.org/>

**Example Segments of Third-Party Code**

* **Text Preprocessing**: Examples of tokenization and padding from TensorFlow and Keras documentation.
* **Early Stopping Implementation**: Adapted from TensorFlow examples for callbacks.
* **Loss and Accuracy Visualization**: Based on examples from Matplotlib tutorials.