# DELE CA1 PART B

Abarna Rajarethinam
DAAA/2B/22

# Importing Modules

```python
# Core libraries
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns

# Text preprocessing
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from deep_translator import GoogleTranslator

# Model development
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import (
    Embedding, LSTM, Dense, Dropout, Bidirectional,
    GlobalMaxPooling1D, BatchNormalization
)
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

# Data utilities
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight

# Evaluation metrics
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    classification_report, roc_curve, auc
)
```

# DATA EXPLORATION

```
print(df.head())        # Preview first few rows
print(df.columns)       # Check column names
print(df.shape)         # See number of rows and columns
```

```
                                              Review  Score  \
0  Filem ini hebat! Aksi yang mendebarkan dan plo...    0.1
1  Filem ini hebat! Aksi yang mendebarkan dan plo...    0.9
2  Filem ini hebat! Aksi yang mendebarkan dan plo...    0.5
3  Sayang sekali, pelakon tidak memberikan persem...    0.8
4  Jalan cerita yang kompleks dan penuh emosi. Su...    0.2

   Are there ways for you to generate more data? Spliting up sentences, would that help?  \
0                                                NaN
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

  Language
0    Malay
1    Malay
2    Malay
3    Malay
4    Malay
Index(['Review', 'Score',
       'Are there ways for you to generate more data? Spliting up sentences, would that help?',
       'Language'],
      dtype='object')
(527, 4)
```

To further inspect the dataset, I used `df.info()` to view the column data types, non-null counts, and memory usage. The dataset contains 527 entries across 4 columns.

- The `'Review'` and `'Language'` columns are complete with 527 non-null entries.
- The `'Score'` column has 2 missing values.
- The column `'Are there ways for you to generate more data? Splitting up sentences, would that help?'` has 0 non-null values, meaning it's entirely empty and should be dropped.

The missing values were confirmed using `df.isnull().sum()`, which provides a clear count of missing entries per column. These insights will guide the next steps in cleaning, particularly removing the empty column and handling the two missing scores.

```
# Basic structure
print(df.info())

# Missing values per column
print("\nMissing values:")
print(df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 527 entries, 0 to 526
Data columns (total 4 columns):
 #   Column                                                                                Non-Null Count  Dtype
---  ------                                                                                --------------  -----
 0   Review                                                                                527 non-null    object
 1   Score                                                                                 525 non-null    float64
 2   Are there ways for you to generate more data? Spliting up sentences, would that help?  0 non-null      float64
 3   Language                                                                              527 non-null    object
dtypes: float64(2), object(2)
memory usage: 16.6+ KB
None

Missing values:
Review                                                                                  0
Score                                                                                   2
Are there ways for you to generate more data? Spliting up sentences, would that help?  527
Language                                                                                0
dtype: int64
```

**Language Distribution**

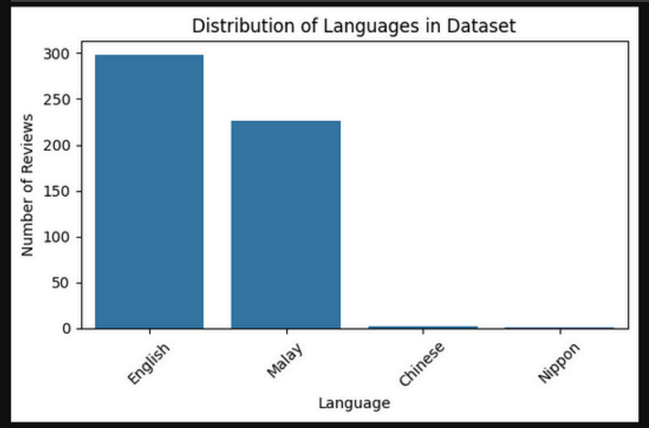
- The dataset has 527 rows and 4 columns
- ‘Review’ and ‘Language’ columns are complete, while ‘Score’ has 2 missing values.
- Data is in multiple languages, and missing values were confirmed using df.info() and df.isnull().sum().

# DATA EXPLORATION

```
print("\nLanguages in dataset:")
print(df['Language'].value_counts())
```

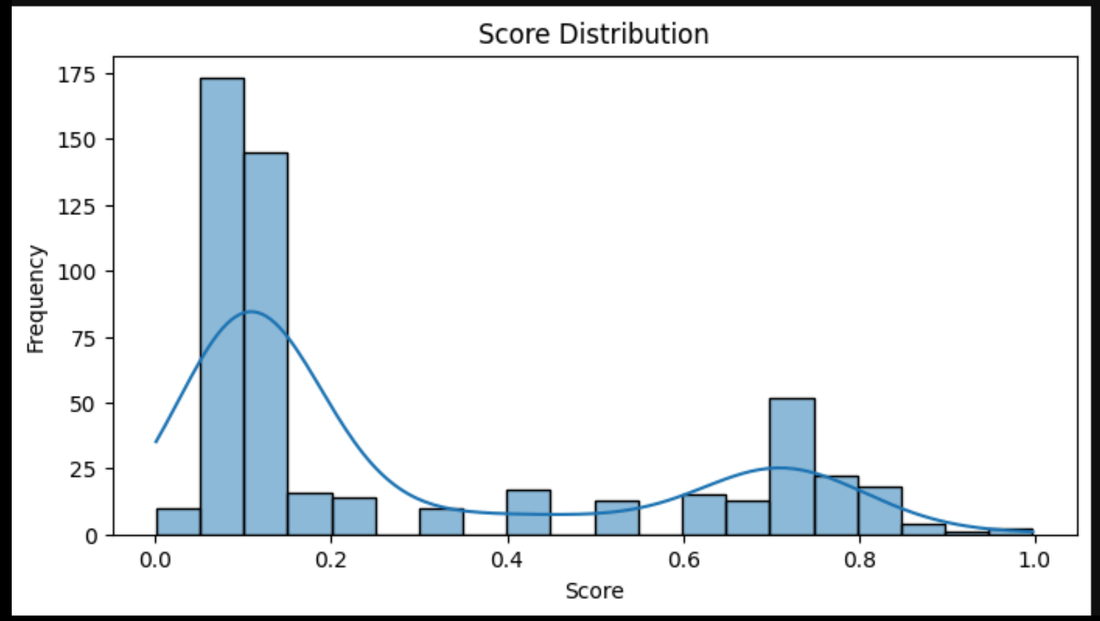
```
Languages in dataset:
Language
English    298
Malay      226
Chinese      2
Nippon       1
Name: count, dtype: int64
```

```
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='Language', order=df['Language'].value_counts().index)
plt.title('Distribution of Languages in Dataset')
plt.xlabel('Language')
plt.ylabel('Number of Reviews')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




- The dataset has 527 rows and 4 columns
- Most reviews are in English (298) and Malay (226), with a few in Chinese (2) and Nippon (1).

```
plt.figure(figsize=(8,4))
sns.histplot(df['Score'], bins=20, kde=True)
plt.title('Score Distribution')
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.show()
```




- The score distribution is imbalanced, with most reviews having low sentiment scores (0.0–0.2) and few high-score reviews.
- This may bias the model toward lower scores, so methods like class weighting or data augmentation may be needed to improve balance.

# DATA CLEANING

```
df.drop(df.columns[2], axis=1, inplace=True)
print(df.head())
```

```
                                              Review  Score Language
0  Filem ini hebat! Aksi yang mendebarkan dan plo...    0.1    Malay
1  Filem ini hebat! Aksi yang mendebarkan dan plo...    0.9    Malay
2  Filem ini hebat! Aksi yang mendebarkan dan plo...    0.5    Malay
3  Sayang sekali, pelakon tidak memberikan persem...    0.8    Malay
4  Jalan cerita yang kompleks dan penuh emosi. Su...    0.2    Malay
```

- The third column was dropped as it was irrelevant to sentiment analysis and helped simplify the dataset by removing non-contributing data.

```
def translate_to_english(text):
    try:
        return GoogleTranslator(source='auto', target='en').translate(text)
    except Exception as e:
        return text  # fallback in case of error

df['Review'] = df['Review'].apply(lambda x: translate_to_english(str(x)))
print(df.head())
```

```
                                              Review  Score
0  A big surprise in the middle of the film! Thri...   0.12
1  A big surprise in the plot! Thrilling action t...   0.10
2  A cinematic experience that is unforgettable. ...   0.10
3  A cinematic marvel! The visuals are breathtaki...   0.08
4  A complex yet engaging plot. A major surprise ...   0.06
```

```
# Count duplicate reviews
num_duplicates = df.duplicated(subset='Review').sum()
print(f"Number of duplicate reviews: {num_duplicates}")
```

```
Number of duplicate reviews: 122
```

```
df = df.drop_duplicates()
```

```
df = df.groupby('Review', as_index=False).agg({'Score': 'mean'})
print(df.head())
```

```
                                              Review  Score
0  A big surprise in the middle of the film! Thri...   0.12
1  A big surprise in the plot! Thrilling action t...   0.10
2  A cinematic experience that is unforgettable. ...   0.10
3  A cinematic marvel! The visuals are breathtaki...   0.08
4  A complex yet engaging plot. A major surprise ...   0.06
```

- 122 duplicate reviews were removed using drop_duplicates(), and groupby() was used to average sentiment scores for repeated texts, reducing redundancy and noise in the data.

- All reviews were translated into English to ensure consistency and compatibility with English-based models, using a try-except block to handle translation errors smoothly.

# DATA CLEANING

```
df['Review'] = df['Review'].str.lower()
print(df.head())
```

```
                                              Review  Score
0  a big surprise in the middle of the film! thri...   0.12
1  a big surprise in the plot! thrilling action t...   0.10
2  a cinematic experience that is unforgettable. ...   0.10
3  a cinematic marvel! the visuals are breathtaki...   0.08
4  a complex yet engaging plot. a major surprise ...   0.06
```

- All review text was converted to lowercase to ensure uniformity, reduce vocabulary size, and improve model generalization and consistency during training.

```
def clean_text(text):
    # Keep only letters, spaces, and fullstops
    text = re.sub(r'[^a-zA-Z\s\.]', '', text)
    return text

df['Review'] = df['Review'].apply(clean_text)
print(df.head())
```

```
                                              Review  Score
0  a big surprise in the middle of the film thril...   0.12
1  a big surprise in the plot thrilling action th...   0.10
2  a cinematic experience that is unforgettable. ...   0.10
3  a cinematic marvel the visuals are breathtakin...   0.08
4  a complex yet engaging plot. a major surprise ...   0.06
```

- Punctuation, numbers, and special characters were removed to clean the text, while full stops were kept to preserve sentence boundaries for future sentence-level analysis or augmentation.

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

def remove_stopwords(text):
    return ' '.join([word for word in text.split() if word not in stop_words])

df['Review'] = df['Review'].apply(remove_stopwords)
print(df.head())
```

```
                                              Review  Score
0          big surprise middle film thrilling action   0.12
1     big surprise plot thrilling action throughout.   0.10
2   cinematic experience unforgettable. im impressed   0.10
3  cinematic marvel visuals breathtaking narrativ...   0.08
4  complex yet engaging plot. major surprise awai...   0.06
```

- Common stopwords (e.g., "the", "is", "and") were removed to reduce noise and help the model focus on sentiment-relevant words, improving its ability to distinguish between positive and negative reviews.

# DATA CLEANING

```python
lemmatizer = WordNetLemmatizer()

def lemmatize_text(text):
    return ' '.join([lemmatizer.lemmatize(word, pos='v') for word in text.split()])

df['Review'] = df['Review'].apply(lemmatize_text)
print(df.head())
```

```
                                              Review  Score
0              big surprise middle film thrill action   0.12
1         big surprise plot thrill action throughout.   0.10
2      cinematic experience unforgettable. im impress   0.10
3  cinematic marvel visuals breathtaking narrativ...   0.08
4  complex yet engage plot. major surprise await end   0.06
```

- A simple lemmatization approach was used by treating all words as verbs, reducing word forms to their base (e.g., “running” → “run”) to lower vocabulary size and improve consistency without complex POS tagging.

# Feature engineering

```python
def score_to_label(score):
    if score <= 0.5:
        return 0  # good
    else:
        return 1  # bad

df['Label'] = df['Score'].apply(score_to_label)
print(df['Label'].value_counts())
print(df.head())
```

```
Label
0    308
1     97
Name: count, dtype: int64
                                              Review  Score  Label
0            big surprise middle film thrill action   0.12      0
1        big surprise plot thrill action throughout.   0.10      0
2     cinematic experience unforgettable. im impress   0.10      0
3  cinematic marvel visuals breathtaking narrativ...   0.08      0
4  complex yet engage plot. major surprise await end   0.06      0
```

- Multi-class and regression approaches were tested but struggled due to class imbalance and interpretability issues.
- A binary classification was chosen, labeling scores ≤ 0.5 as positive (0) and > 0.5 as negative (1), simplifying the task and improving consistency.

# Feature engineering

```python
# Define features and labels
X = df['Review']           # Features (text)
y = df['Label'].astype(int)  # Target (0 or 1)

X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=1, stratify=y_temp)

# Confirm shapes
print(f"Train: {X_train.shape}, Validation: {X_val.shape}, Test: {X_test.shape}")
```

```
Train: (283,), Validation: (61,), Test: (61,)
```

- The dataset was split into training, validation, and test sets

```python
# Initialize lists to hold expanded data
expanded_X_train = []
expanded_y_train = []

# Loop through each review and its label in training data
for review, label in zip(X_train, y_train):
    # Split the review into sentences using full stops
    sentences = [s.strip() for s in review.split('.') if s.strip()]

    # Add each sentence and repeat the label
    expanded_X_train.extend(sentences)
    expanded_y_train.extend([label] * len(sentences))

# Confirm results
print(f"Original training samples: {len(X_train)}")
print(f"Expanded training samples: {len(expanded_X_train)}")
```

```
Original training samples: 283
Expanded training samples: 505
```

- Training reviews were split into individual sentences using full stops to increase data volume and focus on clearer sentiment cues, with each sentence inheriting the original label to ensure consistency.

# Feature engineering

```
# Custom preprocessing pipeline
class TextPreprocessingPipeline:
    def __init__(self, num_words=5000, max_len=500, oov_token="<OOV>"):
        self.tokenizer = Tokenizer(num_words=num_words, oov_token=oov_token)
        self.max_len = max_len

    def fit(self, texts):
        """Learn vocabulary from training data."""
        self.tokenizer.fit_on_texts(texts)

    def transform(self, texts):
        """Convert texts to padded sequences."""
        sequences = self.tokenizer.texts_to_sequences(texts)
        return pad_sequences(sequences, maxlen=self.max_len, padding='post', truncating='post')

    def fit_transform(self, texts):
        """Fit tokenizer and return padded sequences."""
        self.fit(texts)
        return self.transform(texts)

# Initialize the pipeline
pipeline = TextPreprocessingPipeline(num_words=5000, max_len=500)

# Fit and transform training data (sentence-split)
X_train_pad = pipeline.fit_transform(expanded_X_train)

# Transform validation and test sets (not sentence-split)
X_val_pad = pipeline.transform(X_val)
X_test_pad = pipeline.transform(X_test)
expanded_y_train = np.array(expanded_y_train)
```

- A custom text preprocessing pipeline was used to tokenize and pad text, fitting the tokenizer only on the sentence-level training data.
- Text was converted to sequences and padded to a fixed length, while validation and test sets were processed using the same tokenizer without sentence splitting to maintain evaluation fairness.

# Addressing class imbalance

```python
# addressing imbalance
# Compute weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(df['Label']),
    y=df['Label']
)

# Convert to dictionary
class_weights = dict(enumerate(class_weights))
print(class_weights)
```

```
{0: 0.6574675324675324, 1: 2.0876288659793816}
```

- Class weights were computed using compute_class_weight() to give higher weight to the minority class (1) and lower weight to the majority class (0), helping the model focus more on underrepresented sentiment during training.

# Addressing class imbalance

```python
# addressing imbalance
# Compute weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(df['Label']),
    y=df['Label']
)

# Convert to dictionary
class_weights = dict(enumerate(class_weights))
print(class_weights)
```

```
{0: 0.6574675324675324, 1: 2.0876288659793816}
```

- Class weights were computed using compute_class_weight() to give higher weight to the minority class (1) and lower weight to the majority class (0), helping the model focus more on underrepresented sentiment during training.

# Model building and improvement

```
model = Sequential()

# Embedding layer with more expressive word vectors
model.add(Embedding(input_dim=5000, output_dim=128, input_length=500))

# Bidirectional LSTM with return_sequences=True to allow pooling
model.add(Bidirectional(LSTM(64, return_sequences=True)))

# Global max pooling to reduce dimensionality and capture key features
model.add(GlobalMaxPooling1D())

# Dense layer for non-linear transformation
model.add(Dense(64, activation='relu'))

# Dropout to prevent overfitting
model.add(Dropout(0.4))

# Output layer for binary classification
model.add(Dense(1, activation='sigmoid'))


checkpoint = ModelCheckpoint(
    filepath='RNN_best_weights.h5',
    monitor='val_accuracy',
    save_best_only=True,
    save_weights_only=True,
    mode='max',
    verbose=1
)

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=4, min_lr=1e-5, verbose=1)

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

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 500, 128)          640000

 bidirectional_6 (Bidirectio  (None, 500, 128)         98816
 nal)

 global_max_pooling1d_5 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_11 (Dense)            (None, 64)                8256

 dropout_6 (Dropout)         (None, 64)                0

 dense_12 (Dense)            (None, 1)                 65

=================================================================
Total params: 747,137
Trainable params: 747,137
Non-trainable params: 0
_________________________________________________________________
```

- Started with simple models (Embedding + LSTM/Dense), but they only achieved ~70% accuracy due to limited pattern learning.
- Deeper LSTM models improved to ~80% but suffered from overfitting and training instability.
- Final best model achieved >90% test accuracy with a well-balanced architecture:
- Embedding (128 dims): Transforms words into dense vectors, enabling the model to capture semantic relationships.
- Bidirectional LSTM (64 units): Reads text in both directions, improving context understanding.
- GlobalMaxPooling1D: Selects the most important feature across time steps, reducing overfitting and flattening LSTM output.
- Dense layer (64, ReLU): Adds non-linearity to learn deeper, high-level patterns.
- Dropout (0.4): Prevents overfitting by randomly turning off neurons during training.
- Sigmoid Output Layer: Outputs a probability for binary sentiment classification (Good vs. Bad).
- Adam Optimizer: Chosen for its adaptive learning rate and fast convergence, making it ideal for deep NLP models.

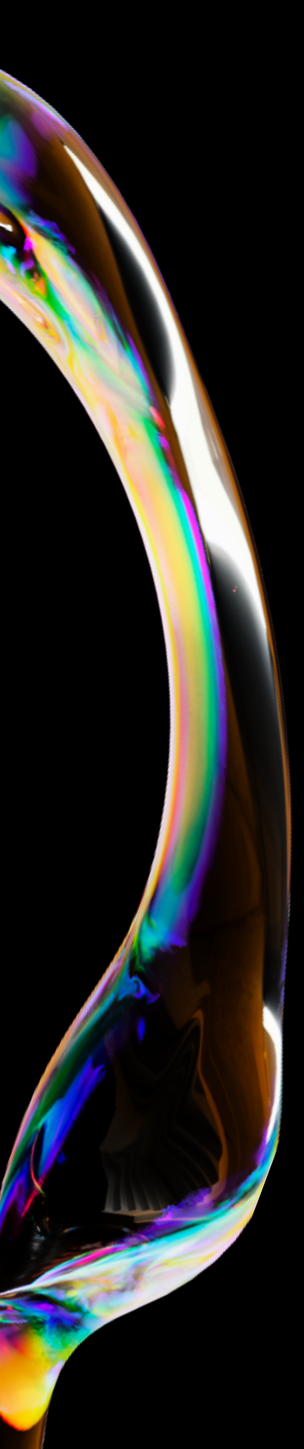
# TRAINING MODEL

```python
# Train the model using sentence-split and padded data
history = model.fit(
    X_train_pad,
    expanded_y_train,
    validation_data=(X_val_pad, y_val),
    epochs=40,
    batch_size=64,
    class_weight=class_weights,
    callbacks=[checkpoint, reduce_lr]
)
```

```
Epoch 37/40
8/8 [==============================] - ETA: 0s - loss: 0.0171 - accuracy: 0.9941
Epoch 37: val_accuracy did not improve from 0.96721
8/8 [==============================] - 3s 384ms/step - loss: 0.0171 - accuracy: 0.9941 - val_loss: 0.1009 - val_accuracy: 0.9508 - lr: 1.5625e-05
Epoch 38/40
8/8 [==============================] - ETA: 0s - loss: 0.0127 - accuracy: 0.9921
Epoch 38: val_accuracy did not improve from 0.96721
8/8 [==============================] - 3s 384ms/step - loss: 0.0127 - accuracy: 0.9921 - val_loss: 0.1010 - val_accuracy: 0.9508 - lr: 1.5625e-05
Epoch 39/40
8/8 [==============================] - ETA: 0s - loss: 0.0155 - accuracy: 0.9941
Epoch 39: val_accuracy did not improve from 0.96721
8/8 [==============================] - 3s 386ms/step - loss: 0.0155 - accuracy: 0.9941 - val_loss: 0.1010 - val_accuracy: 0.9508 - lr: 1.5625e-05
Epoch 40/40
8/8 [==============================] - ETA: 0s - loss: 0.0143 - accuracy: 0.9921
Epoch 40: val_accuracy did not improve from 0.96721

Epoch 40: ReduceLROnPlateau reducing learning rate to 1e-05.
8/8 [==============================] - 3s 388ms/step - loss: 0.0143 - accuracy: 0.9921 - val_loss: 0.1011 - val_accuracy: 0.9508 - lr: 1.5625e-05
```

- Model was trained on padded sequences (X_train_pad) and corresponding binary sentiment labels (expanded_y_train).
- A validation set (X_val_pad, y_val) was used to monitor generalization during training.
- Trained for 40 epochs with a batch size of 64 to balance speed and stability.
- Class weights were applied to reduce bias against underrepresented sentiment classes.
- Used ModelCheckpoint to save the best model based on validation accuracy.
- Used ReduceLROnPlateau to lower the learning rate when validation performance plateaued.
- This setup ensured the model remained robust, well-regularized, and tuned for optimal performance.



# EVALUATING ON TEST DATA

```
# Load the best saved model
model.load_weights('RNN_best_weights.h5')

# Evaluate the best model on the test set
loss, accuracy = model.evaluate(X_test_pad, y_test, verbose=1)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
2/2 [==============================] - 0s 119ms/step - loss: 0.1674 - accuracy: 0.9180
Test Accuracy: 91.80%
```

- After training, the best model weights (saved via ModelCheckpoint) were loaded using model.load_weights().
- This ensures evaluation uses the most optimal version of the model, not the final training state.
- The model was evaluated on unseen test data to assess real-world performance.
- It achieved a strong test accuracy of 91.80%, confirming good generalization.

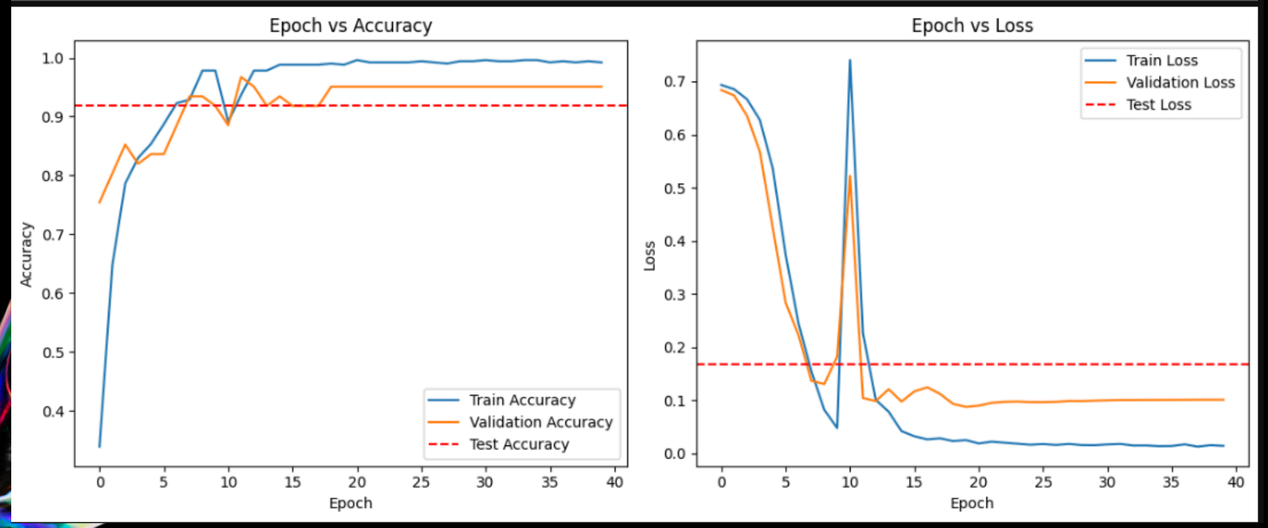
# VISUALISING RESULTS

```python
# Plot Accuracy
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.axhline(y=accuracy, color='r', linestyle='--', label='Test Accuracy')
plt.title('Epoch vs Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.axhline(y=loss, color='r', linestyle='--', label='Test Loss')
plt.title('Epoch vs Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



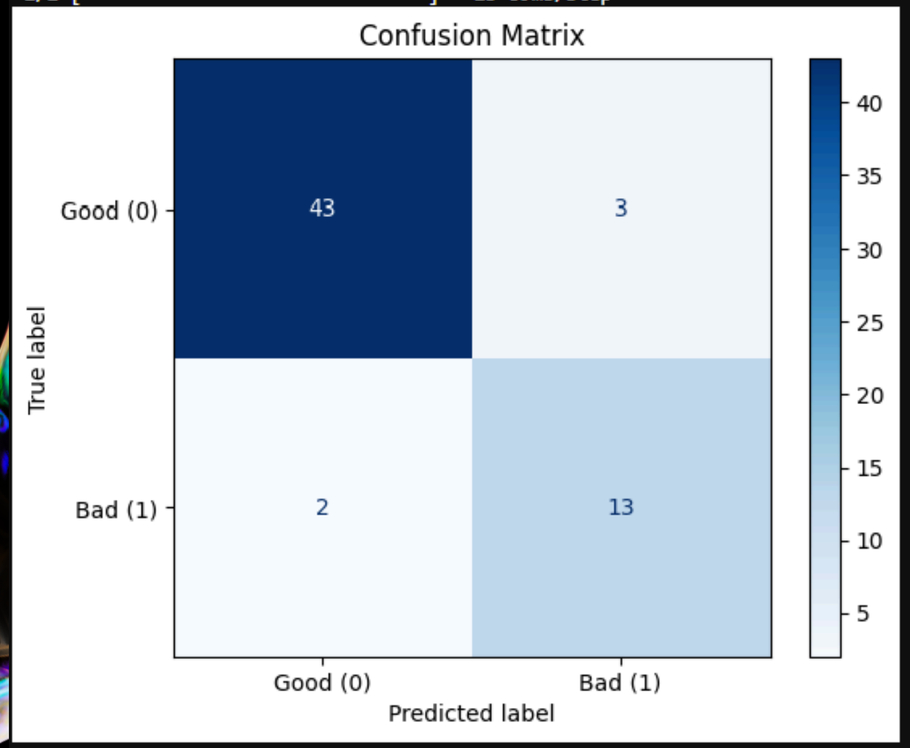

- Accuracy curve shows the model quickly learns and reaches near 100% training accuracy.
- Validation accuracy stabilizes around 93%, showing strong generalization without overfitting.
- Both training and validation loss steadily decrease, indicating effective learning.
- Final test accuracy and loss align well with validation results, confirming consistent performance.
- This validates the model’s robustness, especially under imbalanced, sentence-level input conditions.

# VISUALISING RESULTS

```python
# Get predictions from the model
y_pred_probs = model.predict(X_test_pad)
y_pred = (y_pred_probs > 0.5).astype(int).flatten()  # convert to 0/1

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Good (0)", "Bad (1)"])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

```
2/2 [==============================] - 2s 89ms/step
```




- Confusion matrix shows the model accurately classified 43/46 "Good" and 13/15 "Bad" reviews.
- Only 5 misclassifications occurred, indicating strong performance across both classes.
- Demonstrates that the model doesn’t overfit to the majority class.
- Confirms the effectiveness of class weighting and sentence-level training in improving prediction balance.

# VISUALISING RESULTS

```
print(classification_report(y_test, y_pred, target_names=["Good", "Bad"]))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Good | 0.96 | 0.93 | 0.95 | 46 |
| Bad | 0.81 | 0.87 | 0.84 | 15 |
| accuracy | | | 0.92 | 61 |
| macro avg | 0.88 | 0.90 | 0.89 | 61 |
| weighted avg | 0.92 | 0.92 | 0.92 | 61 |

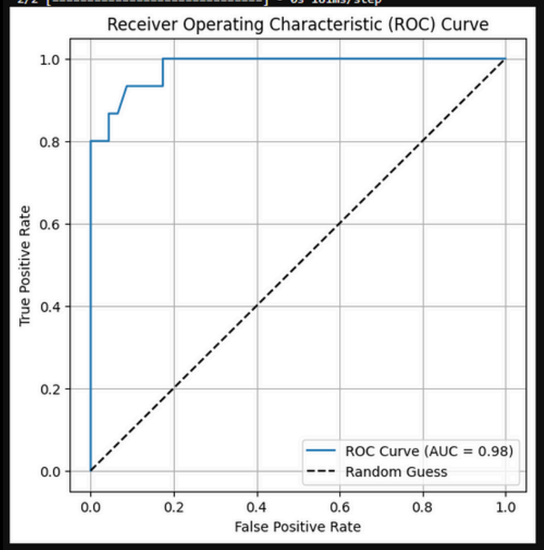
- “Good” class achieved high performance: Precision 0.96, F1-score 0.95, showing accurate positive sentiment detection.
- “Bad” class, though smaller, still performed well: Precision 0.81, Recall 0.87.
- Macro average F1-score of 0.89 and overall accuracy of 92%, indicating balanced classification.
- Confirms model handles both majority and minority classes well, aided by sentence-level augmentation and class weighting.

```
# Get predicted probabilities for the positive class
y_pred_probs = model.predict(X_test_pad).flatten()  # Shape (n_samples,)

# Compute False Positive Rate, True Positive Rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
2/2 [==============================] - 0s 181ms/step
```




- The ROC curve rises steeply toward the top-left, showing excellent true positive vs. false positive trade-off.
- The AUC score is 0.98, indicating strong separability between “Good” and “Bad” sentiment classes.
- Confirms the model is highly reliable and well-calibrated across various threshold settings.