Essay summarization and automated grading system

By,

Sai Charan Merugu

Sai Gottumukkala

Swetha Guntupalli

Karishma Bollineni

Problem Statement

Nowadays, instructorsare often overwhelmed by the amounts of essay correction they need to grade everyday. Traditional methods of evaluation consumes ample amounts of time and heavily relies on manualefforts. Besides, it can also be inconsistent and biased at times.

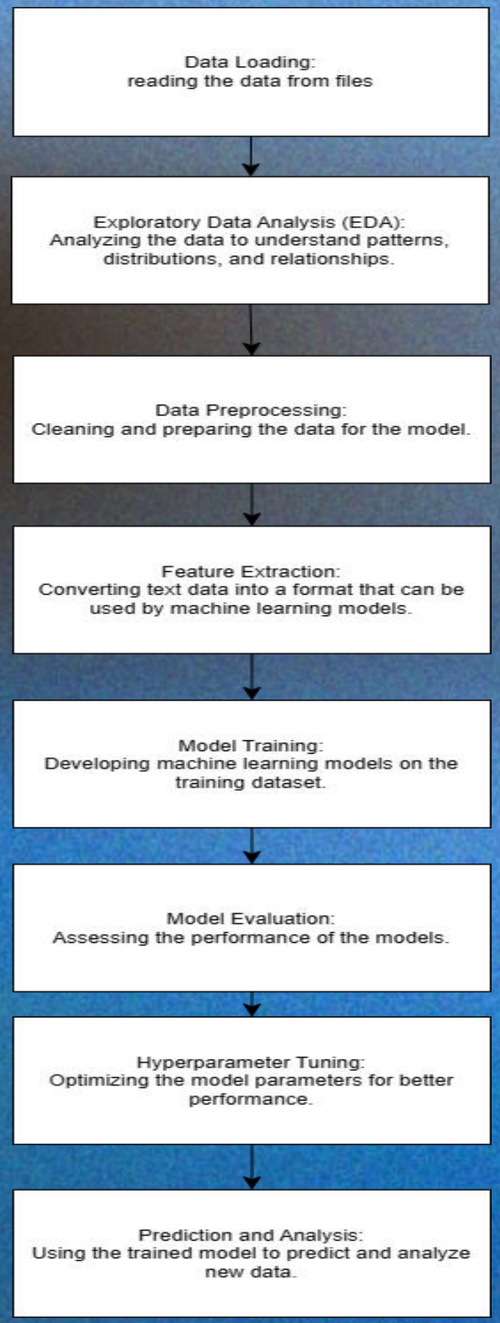
Lack of an efficient system that can do this job instead ofthe actual person, thereby saving some time, triggered us with this idea.

This paved ways for our project idea that can summarize the essay andgrade it accordingly. The summarized essay will serve as a quick review option if the instructor choosesto re-verify the grade assigned.

Model/Methodology

Overall Workflow Diagram

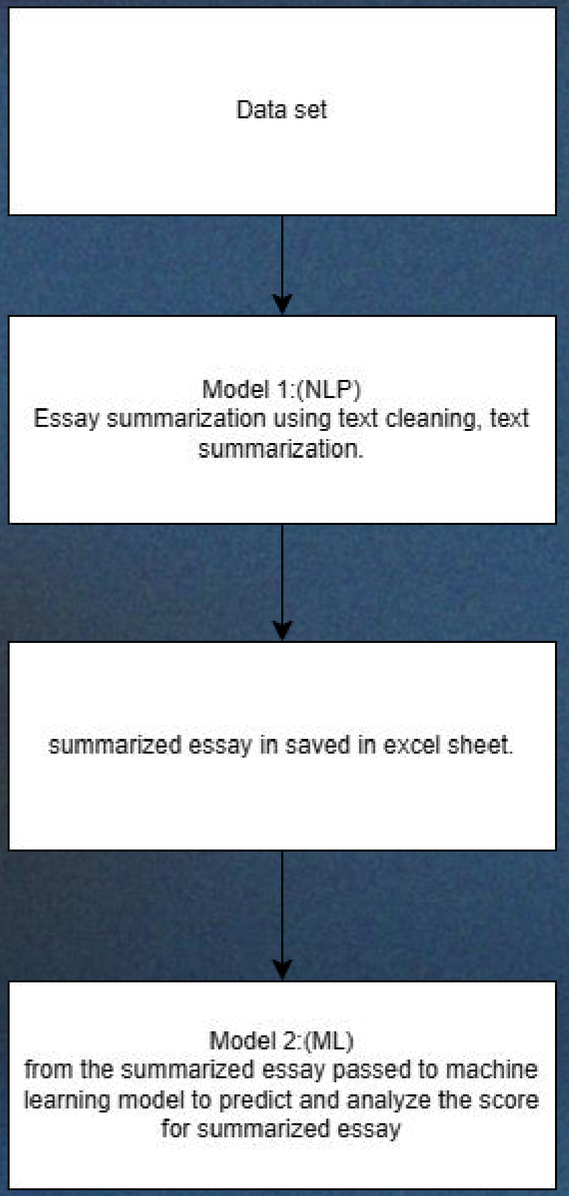
- Data Loading: Reading the data from files.
- Exploratory Data Analysis (EDA): Analyzing the data to understand patterns, distributions, and relationships.
- Data Preprocessing: Cleaning and preparing the data for the model.
- Feature Extraction: Converting text data into a format that can be used by machine learning models.
- Model Training: Developing machine learning models on the training dataset.
- Model Evaluation: Assessing the performance of the models.
- Hyperparameter Tuning: Optimizing the model parameters for better performance.
- Prediction and Analysis: Using the trained model to predict and analyze new data.



Architecture diagram of model

- Model 1:NLP
  - After analyzing the essay ,it undergoes to text cleaning and text summarization. This summarized essay is saved as excel sheet along with score.

- Model 2:Machine Learning:
  - The summarized essay saved from excel under go preprocessing ,then passed to model . The model will learning from the data and predict the score based on the summarized essay.



Dataset

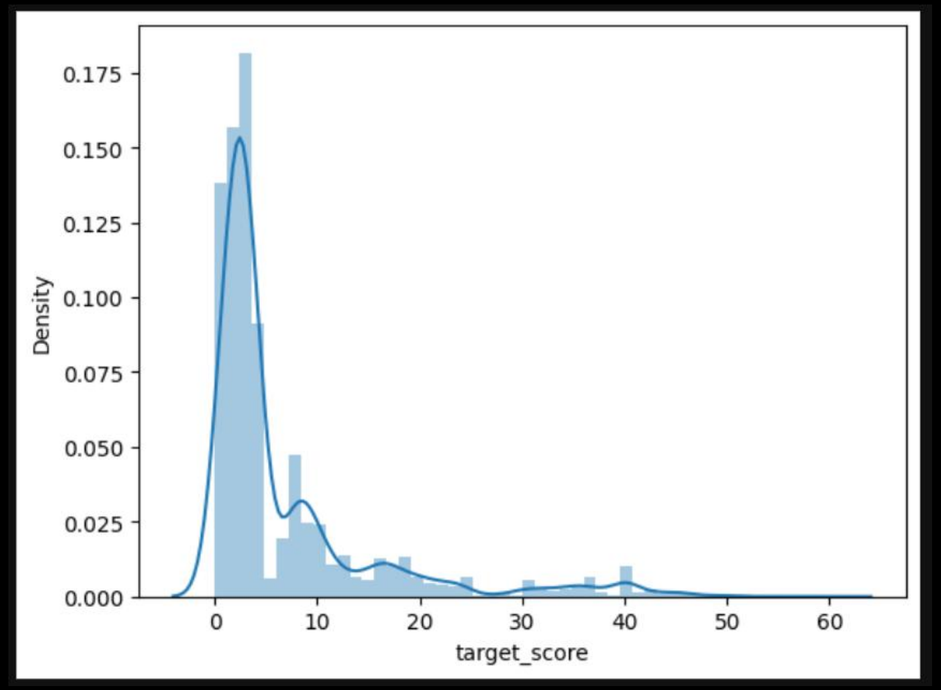
The dataset consist of 8 distinct sets of essays, corresponding to different prompts. each essay vary in length from approximately 150 to 550 words and Some essays are based on provided sources, while others are independent. these responses were crafted by students in grades 7 to 10.Each essay has been hand-graded and has received scores from multiple raters for increased reliability.

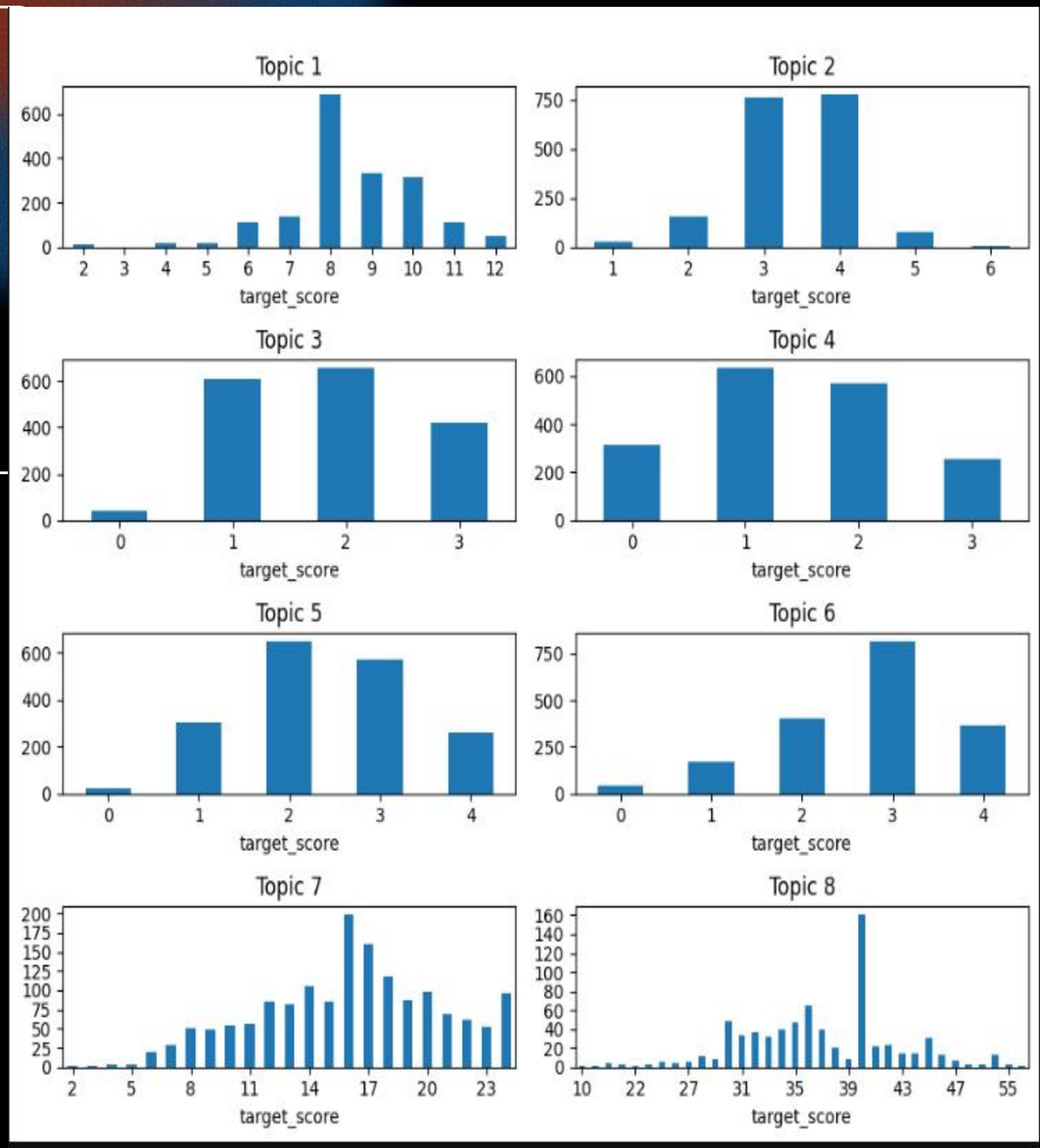
The dataset has following columns:

- Essay_id: A unique identifier for each student essay.
- essay_set: A numerical identifier (1-8) indicating the set to which an essay belongs.
- essay: The ASCII text content of the student's response.
- rater1_domain1: The first rater's score for the primary domain or aspect of the essay.
- rater2_domain1: The second rater's score for the primary domain.
- rater3_domain1: The third rater's score for the primary domain, applicable only for some essays in set 8.
- domain1_score: The final agreed-upon score for the primary domain after resolving differences between raters.
- rater1_domain2, rater2_domain2: Scores from two raters for a secondary domain, applicable only to essays in set 2.
- domain2_score: The resolved score for the secondary domain, only present for essays in set 2.
- Trait Scores (rater1_trait1 to rater3_trait6): Specific characteristic scores for essays in sets 7-8. Each trait score evaluates different aspects of writing, such as organization, vocabulary, grammar, etc

Design of Features/labels with diagram

By comparing the features and concluded useful features from the dataset,categorized the features: essay_id,topic,essay,target_score of the data set ,and plotted the 8 essay topics by target score.
Analyzed the score of the total essays, plot the graph.

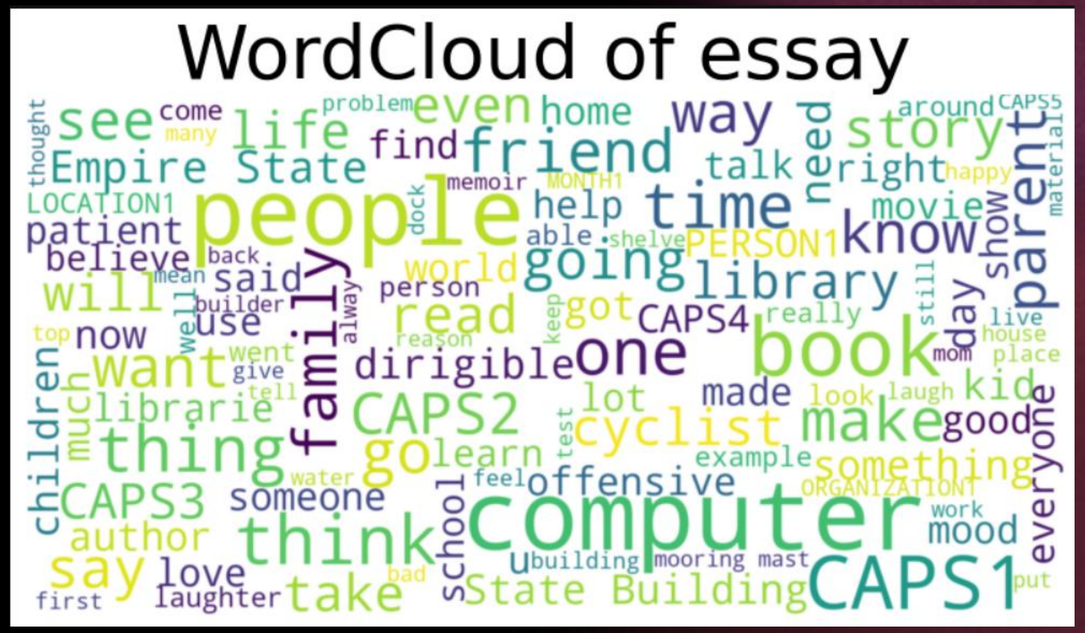


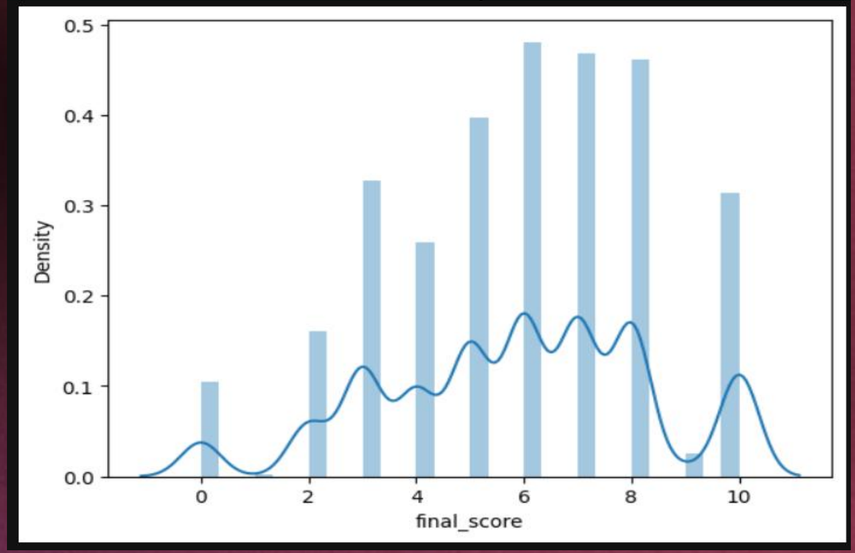


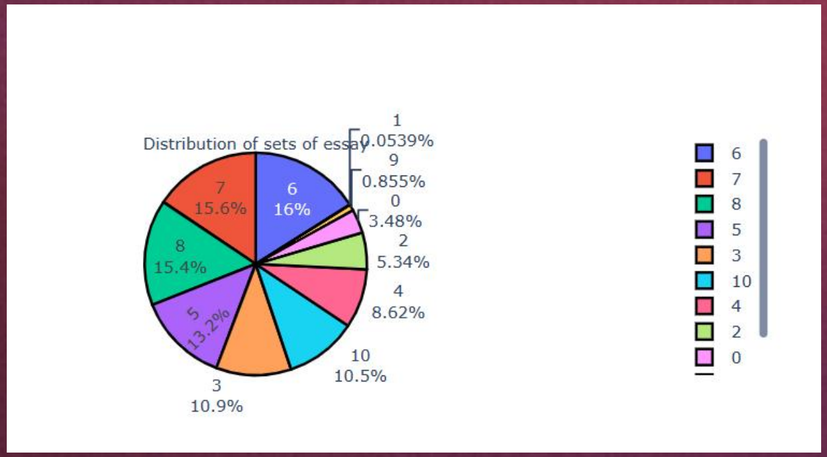
Data Visualization and data preprocessing

Data preprocessing: as the target score is high in some columns and low in some columns, we normalized the and stored in final_score.

```
min_range = [2,1,0,0,0,0,0,0]
max_range = [12,6,3,3,4,4,30,60]

def normalize(x,mi,ma):
    #print("Before Normalization: "+str(x))
    x = (x-mi)/(ma-mi)
    #print("After Normalization : "+str(x))
    return round(x*10)

df['final_score']=df.apply(lambda x:normalize(x['target_score'],min_range[x['topic']-1],max_range[x['topic']-1]),axis=1)
```







Data Visualization and data preprocessing

Data preprocessing: data cleaning for the given essay using regular expression clean html tags, special symbols, slashes, on word punctuation ,numbers, white spaces and convert them to lower case.

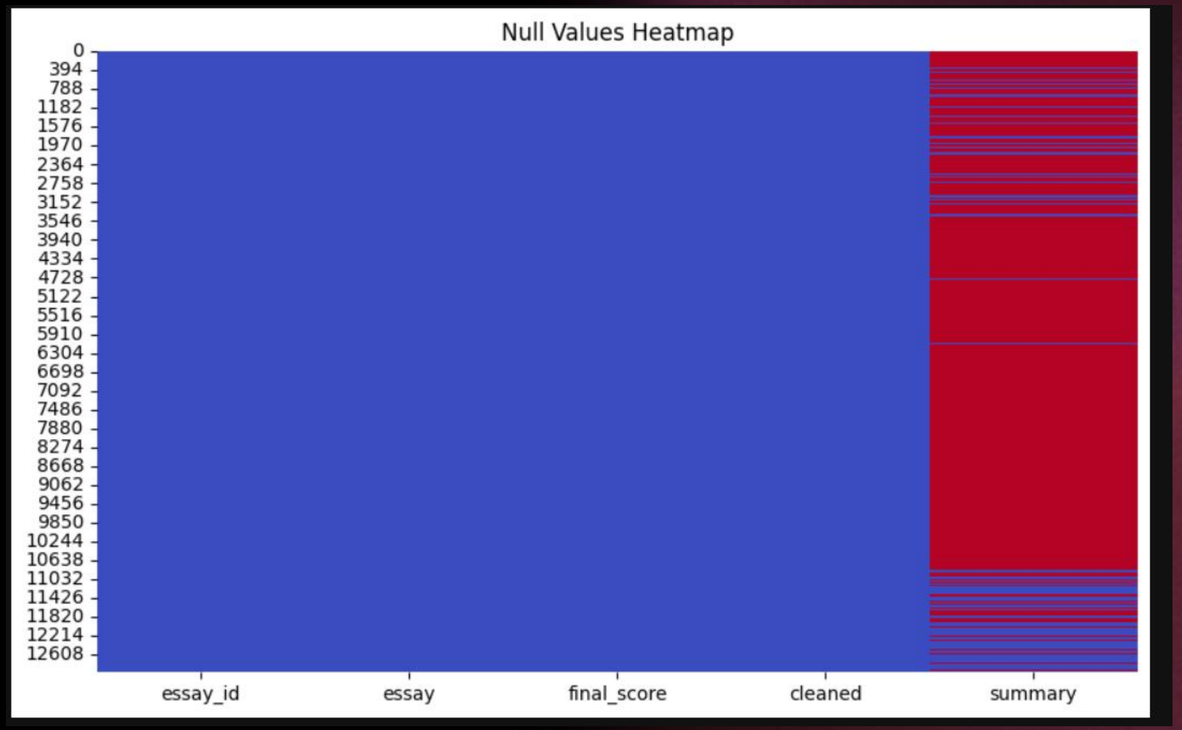
For the clean text in the essay ,we process the text summarization from the learning concept of the course using spacy library , using stop words and punctuation cleaned further text and stored in summary field and exported the excel file.

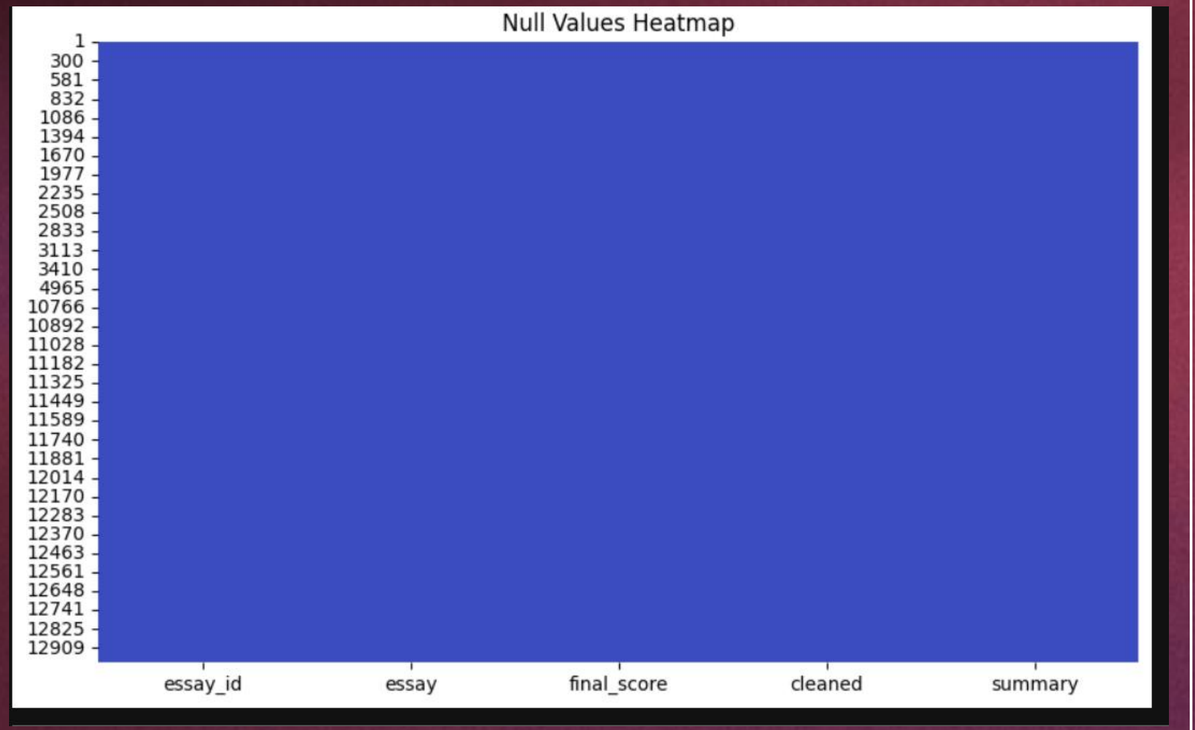
```python
import re

def clean_text(text):
    # Remove placeholders like @CAPS1
    text = re.sub(r'@\w+', '', text)
    # Remove HTML tags if present
    text = re.sub(r'<.*?>', '', text)
    # Handle specific cases like 'skills/affects'
    # Option 1: Replace slashes with spaces to separate words
    text = re.sub(r'/', ' ', text)
    # Option 2: If you prefer to keep words joined post-slash removal, uncomment the following line
    text = re.sub(r'/', '', text)
    # Replace non-word characters (punctuation) with spaces except for apostrophes
    text = re.sub(r'[^\w\s\']', ' ', text)
    # Remove numbers if they are not relevant
    text = re.sub(r'\d+', '', text)
    # Remove any underscores - often used as placeholders
    text = re.sub(r'_', ' ', text)
    # Normalize white space to a single space
    text = re.sub(r'\s+', ' ', text)
    # Convert to lowercase to maintain uniformity
    text = text.lower().strip()
    return text
```

```python
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
import pandas as pd
from heapq import nlargest

# Initialize spaCy
nlp = spacy.load('en_core_web_sm')
stopwords = list(STOP_WORDS)
punctuation += '\n'  # Append newline to punctuation

def summarize_essay(essay, select_length=0.3):
    """ Summarize the essay using spaCy for NLP processing. """
    doc = nlp(essay)
    word_frequencies = {}

    # Calculate word frequencies, ignoring stopwords and punctuation
    for word in doc:
        if word.text.lower() not in stopwords and word.text.lower() not in punctuation:
            word_frequencies[word.text] = word_frequencies.get(word.text, 0) + 1

    # Normalize frequencies and compute sentence scores
    max_frequency = max(word_frequencies.values(), default=1)
    sentence_scores = {}
    for sent in doc.sents:
        for word in sent:
            if word.text.lower() in word_frequencies:
                sentence_scores[sent] = sentence_scores.get(sent, 0) + word_frequencies[word.text.lower()]

    # Select the top sentences based on their scores
    summary_sentences = nlargest(int(len(list(doc.sents)) * select_length), sentence_scores, key=sentence_scores.get)
    summary = ' '.join(sent.text for sent in summary_sentences)

    return summary

def process_and_save_summaries(input_file, output_file_path):
    """ Read essays from an Excel file, summarize them, and save the summaries to a new Excel file. """
    essay_data = input_file
    essay_data['summary'] = essay_data['cleaned'].apply(summarize_essay)

    # Save the updated DataFrame to a new Excel file
    essay_data.to_excel(output_file_path, index=False)
    print(f"Summaries saved to {output_file_path}")

# Specify file paths
input_file_path = df # Path to the input Excel file with essays
output_file_path = r'C:\Users\merug\Desktop\nlp_project\summarized_essays.xlsx'  # Path for the output Excel file with summaries

# Process essays and save summaries
process_and_save_summaries(input_file_path, output_file_path)
```

Data Visualization and data preprocessing

Data preprocessing: after summarizing the essay,before traing checking the null fields and cleaning it. Here are the heatmap representation of the analysis.





Implementation

- NLP libraries
- spaCy: It is anopen-source NLPlibrary designedfor performing complex NLP tasks with ease. spaCy is knowfor its speed and efficiencyand comes with pre-built statistical models and word vectors.
- Regular Expression (re):Regular expressions are used forstring searching and manipulation, oftenfor complex patterns of text. For instance, you mightuse regular expressions tovalidate email addresses, removeunwanted characters, find all URLs in a text, orperformsubstitutionsin a string.
- String:It's oftenused intext processing toremove punctuation from textbeforeprocessing or tokenizing because punctuationmarks caninterfere with how text is interpreted or analyzed.
- Common Librarie
- Numpy:It's heavilyused in numericalcomputations, data transformation, and is a key libraryin the ecosystem of scientific computing inPython.
- Pandas: It providesdata structures likeDataFrames and Series, which are ideal for working withstructureddata. Pandas is widelyused for data processing, cleaning, andanalysis.
- Seaborn: It offers a more powerful and simpler interfacefor creatingcomplex visualizations, including various plot types like scatter plots, barcharts, and violinplots, with sensibledefault designs.
- matplotlib.pyplot: It's used for creating static, interactiveand animated visualizations in Python.

Model 2:Libraries ,algorithm and implementation

Logistic regression:
Logistic Regression is astatistical algorithm usedprimarily forbinary classification problems, whichis a type of predictiveanalysis. Regressionmodels the probabilities for classification probles with two possibleoutcomes. It’s an exension of the linearregression model for classification tasks.
Eg:"Spam" or "NotSpam" in email filteringsystems.
"Maligant" or "Benin" for a tumor diagnosis.

Weused this algorithm to analyzethe summarized essay for classification of scorebased on essay. And used metrics for testing modellearning:Accuracy, Precision ,Recall ,F1 Score,confusionmatrix

Model 2:Libraries ,algorithm and implementation

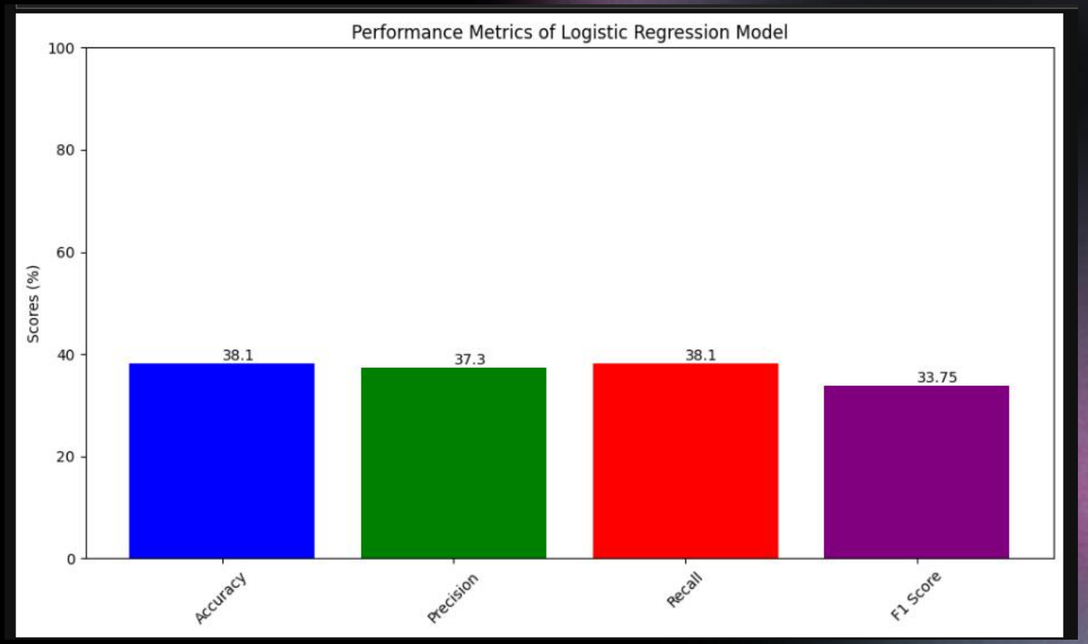
Logistic regression:

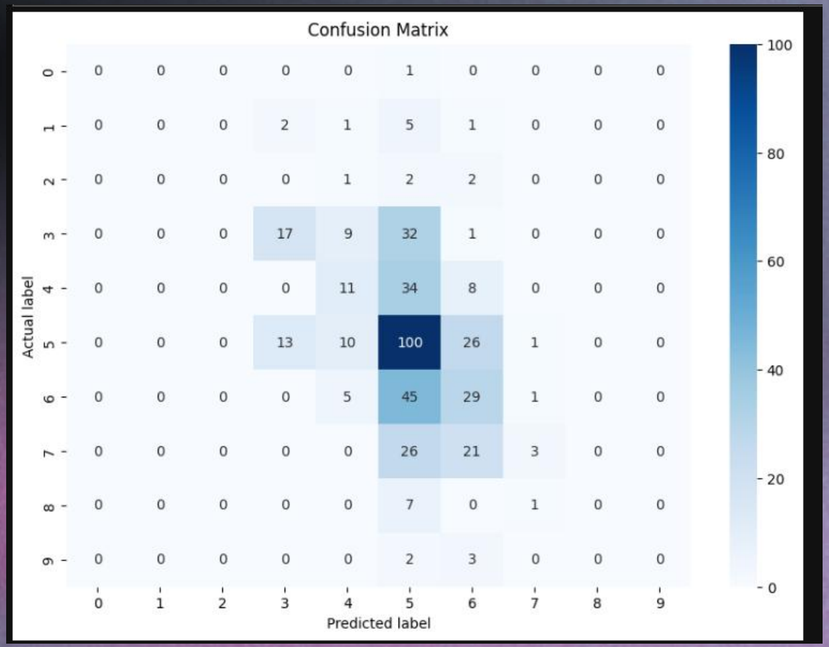
```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
import pandas as pd

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['summary'], df['final_score'], test_size=0.2, shuffle=True, random_state=26)
# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer()
# Fit the vectorizer on the training essays
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
# Initialize and train the Logistic Regression model
logreg_model = LogisticRegression()
logreg_model.fit(X_train_tfidf, y_train)
# Predictions
predictions = logreg_model.predict(X_test_tfidf)
# Evaluation metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='weighted')
recall = recall_score(y_test, predictions, average='weighted')
f1 = f1_score(y_test, predictions, average='weighted')
confusion = confusion_matrix(y_test, predictions)
classification_rep = classification_report(y_test, predictions)
# Print evaluation metrics
print("Logistic Regression Model (Without Hyperparameter Tuning)")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_rep)
```

```
Logistic Regression Model (Without Hyperparameter Tuning)
Accuracy: 0.38095238095238093
Precision: 0.37297650086071066
Recall: 0.38095238095238093
F1 Score: 0.33749662648382217
Confusion Matrix:
 [[ 0  0  0  0  0  1  0  0  0  0]
 [ 0  0  0  2  1  5  1  0  0  0]
 [ 0  0  0  0  1  2  2  0  0  0]
 [ 0  0  0 17  9 32  1  0  0  0]
 [ 0  0  0  0 11 34  8  0  0  0]
 [ 0  0  0 13 10 100 26  1  0  0]
 [ 0  0  0  0  5 45 29  1  0  0]
 [ 0  0  0  0  0 26 21  3  0  0]
 [ 0  0  0  0  0  7  0  1  0  0]
 [ 0  0  0  0  0  2  3  0  0  0]]
```

Model 2:Libraries ,algorithm and implementation

Logistic regression:





Model 2:Libraries ,algorithm and implementation

Logistic regression:we divided the dataset in 80 for training and testing.
fidfVectorizer from sklearn.feature_extraction.text is usedto convert thetext data into numericalfeatures suitable formodel training.

A LogisticRegression model is instantiatedand trained on the vectorized training data (X_train_tfidf).
the trained model is used to predict the scores of thevectorized test data (X_test_tfidf).

Accuracy: The proportion of total predictions that were correct.
Precision (Weighed): The ability ofthe classifier not to label anegative sample a positive, with consideraton for the imbalance in class distribution.
Recall (Weighed): Theability of the clasifier to find all the postive samples, again weighted for class imbalance.
F1 Score (Weighed): A weighted average of precisionand recall, which balances both metrics.
A confusion matrix isgenerated, showing true positives, falsepositives, truenegatives, and falsenegatives.

Decision tree

Decision Tree:
Decision Trees are a type of SupervisedMachine Learning algorithmthat is predominantly used forclassification problems, but can also be adapted for regression.

A Decision Tree algorithmsegments the dataset into branches toform a treestructure. Each internalnode of the treerepresents a test on anattribute (e.g., whether a coinflip comes up heads or tails), each branch represents the outcome of the test, and each leafnode represents a classlabel (decision taken aftercomputing all attributes)

We used this algorithm to analyze thesummarized essay for classification of scorebased on essay. And usedmetrics for testing model learning:Accuracy, Precision ,Recall ,F1 Score,confusion matrix

Decision tree

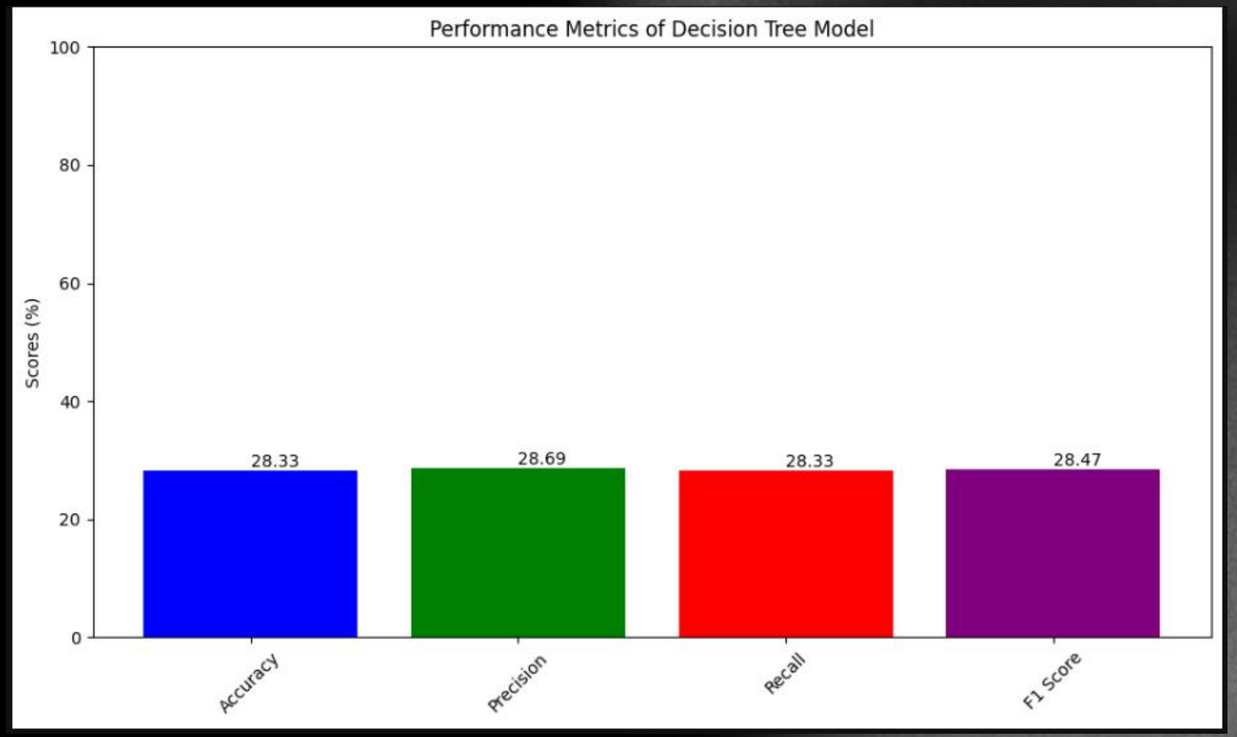
Decision Tree:

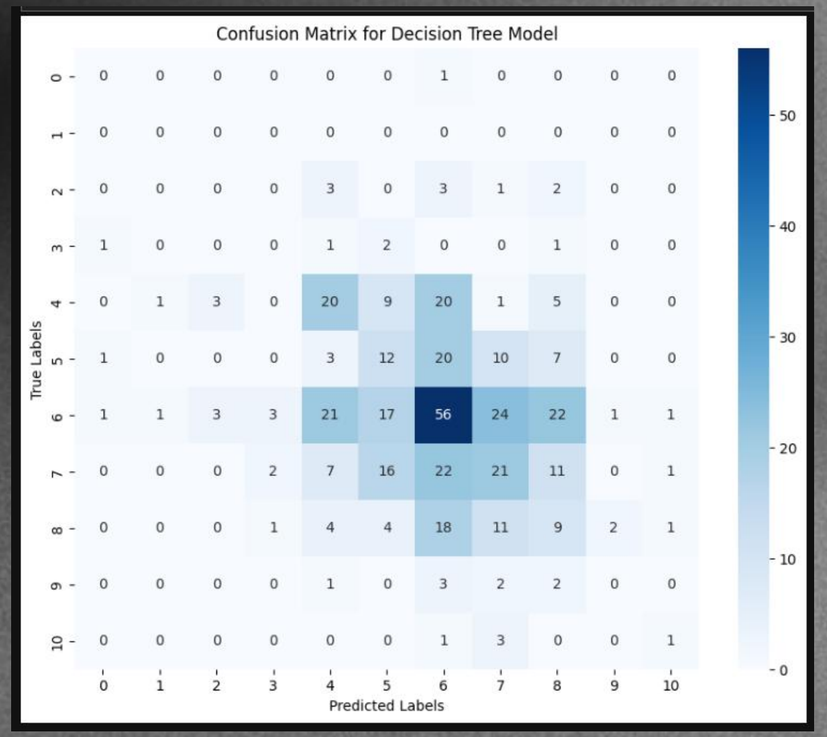
```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, classification_report
import pandas as pd

# Load the dataset containing essays and summaries


# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit the vectorizer on the training essays
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Initialize and train the Decision Tree model
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train_tfidf, y_train)

# Predictions
predictions = dt_model.predict(X_test_tfidf)

# Evaluation metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='weighted')
recall = recall_score(y_test, predictions, average='weighted')
f1 = f1_score(y_test, predictions, average='weighted')
# AUC-ROC score
# try:
#     auc_roc = roc_auc_score(y_test, predictions, average='weighted', multi_class='ovr')
# except ValueError:
#     auc_roc = 0.0  # Set to 0 if AUC-ROC score calculation fails

confusion = confusion_matrix(y_test, predictions)
classification_rep = classification_report(y_test, predictions)

# Print evaluation metrics
print("Decision Tree Model (Without Hyperparameter Tuning)")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
# print("AUC-ROC Score:", auc_roc)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_rep)
```

```
Decision Tree Model (Without Hyperparameter Tuning)
Accuracy: 0.2833333333333333
Precision: 0.28688289827191615
Recall: 0.2833333333333333
F1 Score: 0.28466715057830844
Confusion Matrix:
 [[ 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  3  0  3  1  2  0  0]
 [ 1  0  0  0  1  2  0  0  1  0  0]
 [ 0  1  3  0 20  9 20  1  5  0  0]
 [ 1  0  0  0  3 12 20 10  7  0  0]
 [ 1  1  3  3 21 17 56 24 22  1  1]
 [ 0  0  0  2  7 16 22 21 11  0  1]
 [ 0  0  0  1  4  4 18 11  9  2  1]
 [ 0  0  0  0  1  0  3  2  2  0  0]
 [ 0  0  0  0  0  0  1  3  0  0  1]]
```

Decision tree

Decision Tree:





Decision tree

Decision Tree:
we divided the dataset in 80 for training and testing.
fidfVectorizer from sklearn.feature_extraction.text is used toconvert the textdata into numericalfeatures suitable for modeltraining.

A DecisionTree model is instantiated and trained on the vectorizedtraining data (X_train_tfidf).
the trainedmodel is used to predict the scores of the vectorizedtestdata (X_test_tfidf).

Accuracy: The proportion of totalpredictions that were correct.
Precision (Weighed): The ability of the classifiernot to label a negativesample as positive, with consideration for theimbalance in classdistribution.
Recall (Weighed): The ability of the classifier to find all the positivesamples, again weighed for class imbalance.
F1 Score (Weighed): A weighed average of precision andrecall, which balance both metrics.
A confusion matrix is generated, showingtrue positives, falsepositives, truenegatives, and falsenegatives.

Neural Network

A neuralnetwokmodel is a comptationalframework inspired by the biologicalneralnetworks that constitute anmalbrains. It is capable of learning from obervationaldata and has become a cornerstone for manyapplications in artificialintelligence, including image andspeech recognition, naturllanguageprocessing, and roboticcontrol.

- Basic Struture:
- Neurons: The fundamental units of a neuralnetwork are artificianeurons or nodes. Each neuron receivs input, procsses it, and generates an output.
- Layers: Neurons are rgnizedlayers. There are theetypes of layers in a typicalneuralnetwork:
- Input Layer: Receives the initialdata.
- Hidden Layers: Intermediate lyers that process inputsreceived from previouslayers using weights, biases, and activatiofunctions.
- Output Layer: Produces the finalotput ofthe moel.

Neural Network

```python
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
X_train, X_test, y_train, y_test = train_test_split(df['summary'], df['final_score'].astype(int), test_size=0.2, shuffle=True, random_state=26)
# Example DataFrame loading
```

```python
tfidf = TfidfVectorizer(max_features=10000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```python
import tensorflow as tf

# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_shape=(X_train_tfidf.shape[1],)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(11, activation='softmax')  # Output layer for 11 classes
])

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

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 512)	5,120,512
dropout_4 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 256)	131,328
dropout_5 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 11)	2,827

Total params: 5,254,667 (20.04 MB)
Trainable params: 5,254,667 (20.04 MB)
Non-trainable params: 0 (0.00 B)

```python
history = model.fit(X_train_tfidf, y_train, validation_split=0.1, epochs=10, batch_size=64)
```

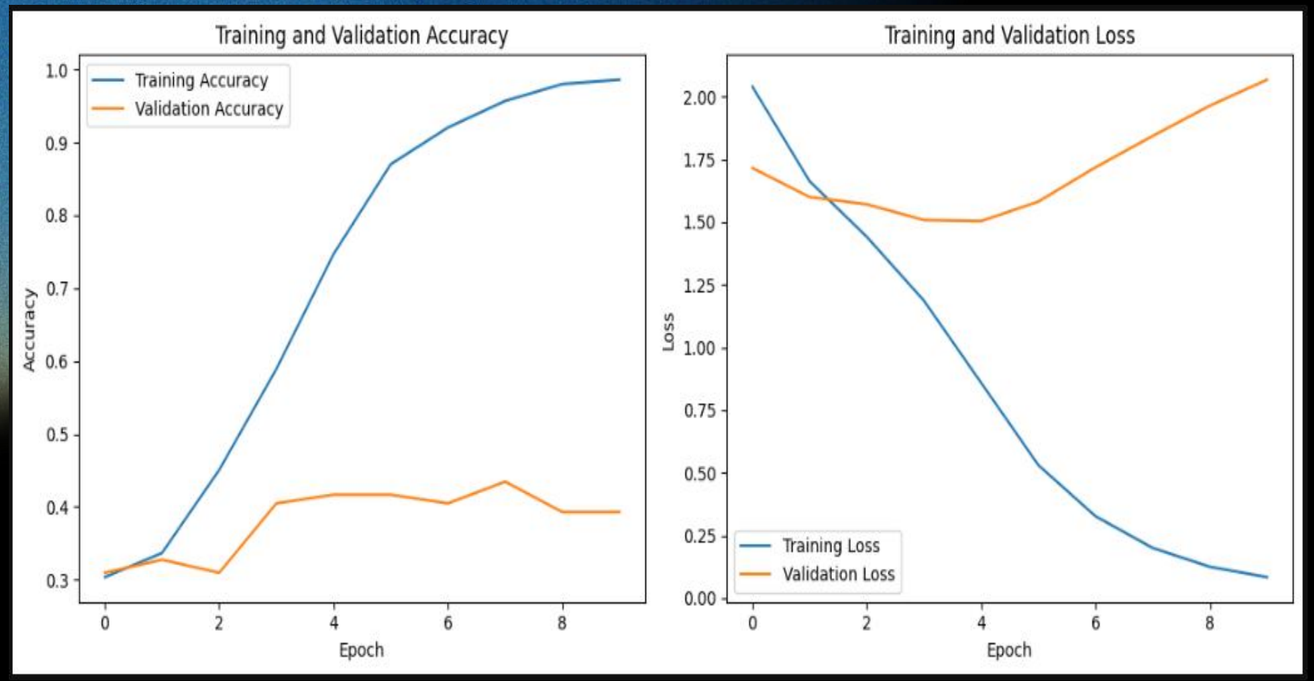
```
Epoch 1/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.2418 - loss: 2.2163 - val_accuracy: 0.3095 - val_loss: 1.7154
Epoch 2/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.3315 - loss: 1.6915 - val_accuracy: 0.3274 - val_loss: 1.5997
Epoch 3/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.4395 - loss: 1.4601 - val_accuracy: 0.3095 - val_loss: 1.5707
Epoch 4/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.5683 - loss: 1.2169 - val_accuracy: 0.4048 - val_loss: 1.5079
Epoch 5/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.7308 - loss: 0.8991 - val_accuracy: 0.4167 - val_loss: 1.5037
Epoch 6/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.8646 - loss: 0.5545 - val_accuracy: 0.4167 - val_loss: 1.5814
Epoch 7/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9253 - loss: 0.3370 - val_accuracy: 0.4048 - val_loss: 1.7179
Epoch 8/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9493 - loss: 0.2261 - val_accuracy: 0.4345 - val_loss: 1.8432
Epoch 9/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9751 - loss: 0.1378 - val_accuracy: 0.3929 - val_loss: 1.9640
Epoch 10/10
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9863 - loss: 0.0898 - val_accuracy: 0.3929 - val_loss: 2.0674
```

```python
loss, accuracy = model.evaluate(X_test_tfidf, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3723 - loss: 2.0819
Test Accuracy: 35.71%
```

```python
import matplotlib.pyplot as plt

# Ensure the figure is large enough to hold both plots with clear visibility
plt.figure(figsize=(12, 5))

# Plot training and validation accuracy
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

Neural Network




```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import numpy as np
```

```python
# Predicting the probabilities
y_pred_probs = model.predict(X_test_tfidf)
# Converting probabilities to class labels
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

```python
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

```
Accuracy: 0.36
Precision: 0.33
Recall: 0.36
F1 Score: 0.34
```

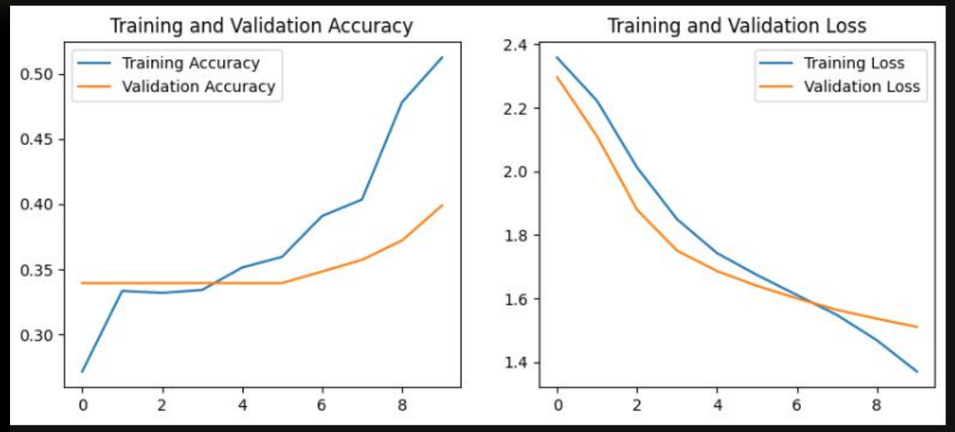
After hyperparameter tuning: neural network

```
import tensorflow as tf
from kerastuner.tuners import RandomSearch

def build_model(hp):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(
        units=hp.Int('units1', min_value=256, max_value=512, step=128),
        activation='relu',
        input_shape=(X_train_tfidf.shape[1],)
    ))
    model.add(tf.keras.layers.Dropout(rate=hp.Float('dropout1', min_value=0.4, max_value=0.6, step=0.1)))
    model.add(tf.keras.layers.Dense(
        units=hp.Int('units2', min_value=128, max_value=256, step=64),
        activation='relu'
    ))
    model.add(tf.keras.layers.Dropout(rate=hp.Float('dropout2', min_value=0.4, max_value=0.6, step=0.1)))
    model.add(tf.keras.layers.Dense(11, activation='softmax'))

    hp_learning_rate = hp.Float('learning_rate', min_value=1e-4, max_value=1e-2, sampling='LOG')
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=hp_learning_rate),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

```
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=1,
    directory='my_dir',
    project_name='keras_tuner_demo'
)

# Start hyperparameter search
tuner.search(X_train_tfidf, y_train, epochs=5, validation_split=0.1)

# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"""
The best number of units in the first densely-connected layer is {best_hps.get('units1')} and the best dropout rate is {best_hps.get('dropout1')}.
```

After hyperparameter tuning: neural network

```python
import matplotlib.pyplot as plt

# Plot accuracy
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.show()
```




```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import numpy as np
```

```python
# Predicting the probabilities
y_pred_probs = model.predict(X_test_tfidf)
# Converting probabilities to class labels
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

```python
# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

```
Accuracy: 0.37
Precision: 0.28
Recall: 0.37
F1 Score: 0.29
```

After hyperparameter tuning: neural network

- Dense Layers: Fully connectedlayers with a variablenumber of units (units1 and units2), both using ReLU activationfunctions. The number of units for theselayers is chosenbased on the hyperparameter search spacedefined for units1and units2.
- Dropout Layers: Used to prevent overfittingby randomlysetting a fractionof input units to 0 duringtraining. Dropout rates (dropout1 and dropout2) are alsovariable.
- Output Layer: A softmaxlayer suitable for multi-classclassification with 11 outputclasses.
- Optimization and Compilation:
- The learningrate for the Adamoptimizer is varied within a specified range using a logarithmic scale (sampling='LOG'), which is effective forexploring orders ofmagnitude.
- The model uses sparse_categorical_crossentropy as the lossfunction, which is typical for multi-class classificationtasks where labels areprovided as integers.
- Kera's Tuner Configuration:
- Random Search: A tunerthat performs a randomsearch on hyperparameters, optimizingfor val_accuracy.
- The search evaluates 10 differentconfigurations (max_trials=10) with each configuration run once (executions_per_trial=1).
- The directory and projectname specifywhere trial data isstored

After hyperparameter tuning: Logistic Regression

logistic regression

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['newton-cg', 'lbfgs', 'liblinear'],
    'penalty': ['l2']
}

# Ensure the model uses multi_class if it's a multi-class classification
logreg = LogisticRegression(max_iter=1000, multi_class='auto')

# Setup the grid search
grid_search = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5, verbose=1, scoring='accuracy', n_jobs=-1)
```

```
# Fit GridSearchCV
grid_search.fit(X_train_tfidf, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

GridSearchCV

estimator: LogisticRegression

LogisticRegression

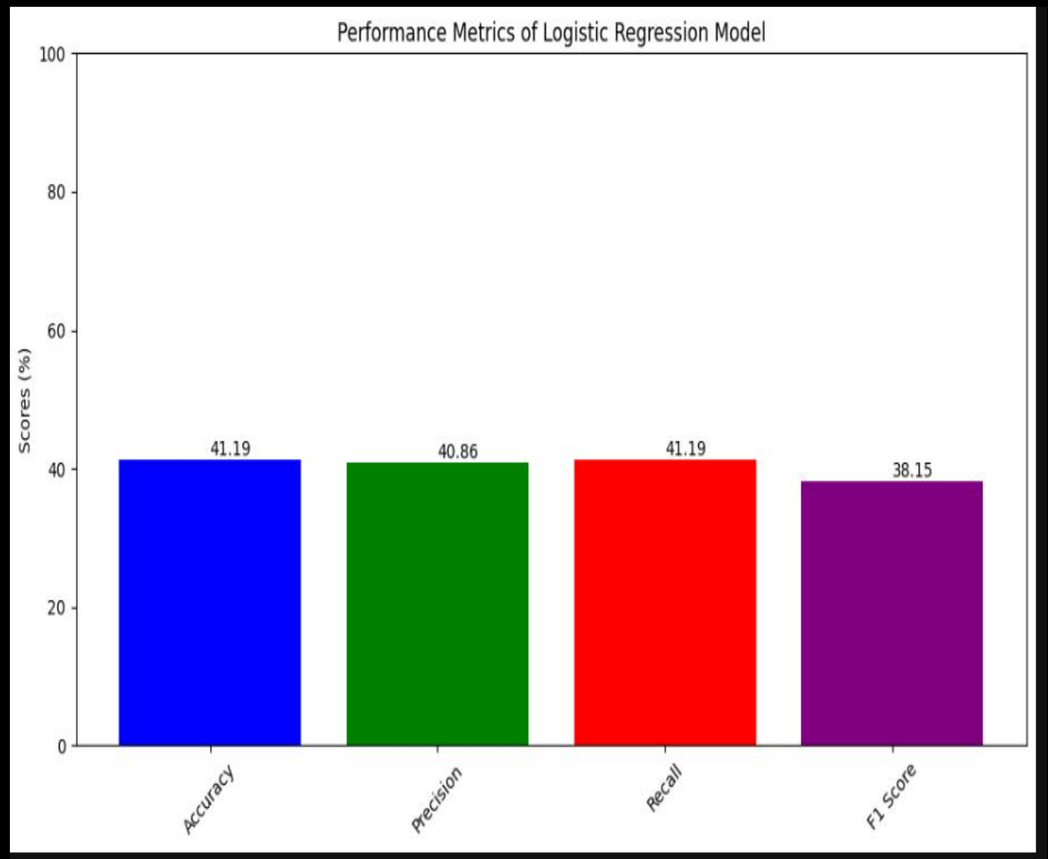
```
# Best parameters
best_params = grid_search.best_params_
print("Best parameters found:", best_params)

# Best model
best_model = grid_search.best_estimator_
```

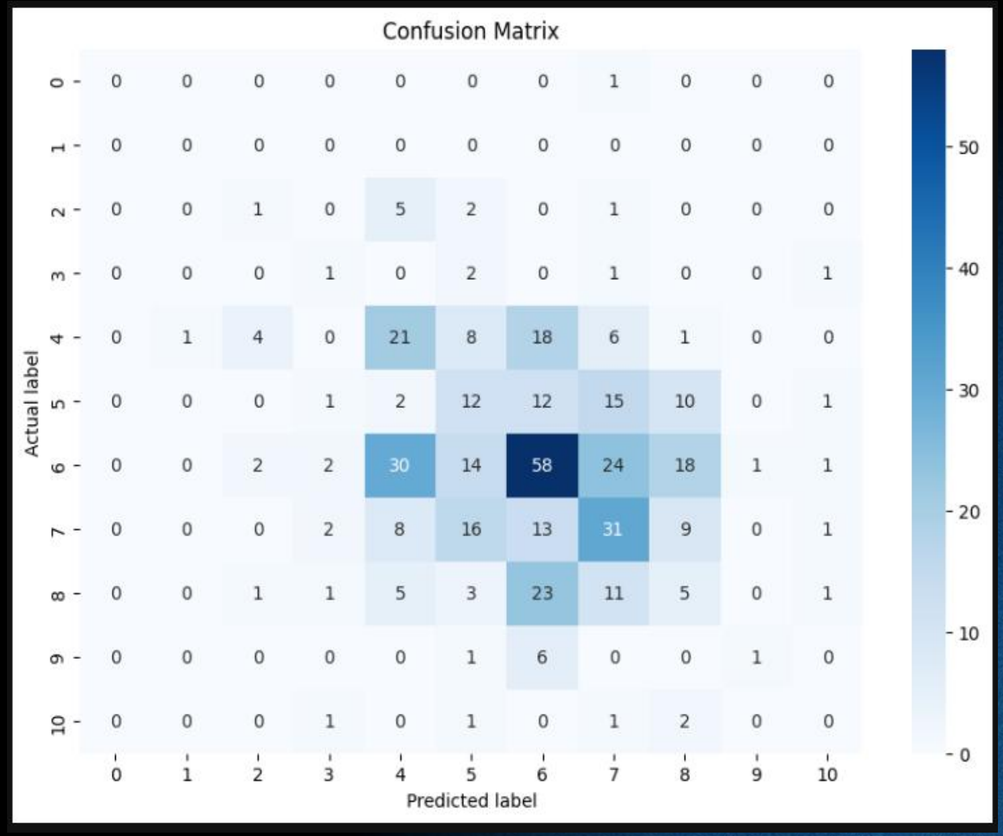
```
Best parameters found: {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
```

```
# Predictions with the best model
best_predictions = best_model.predict(X_test_tfidf)

# Evaluation metrics
best_accuracy = accuracy_score(y_test, best_predictions)
best_precision = precision_score(y_test, best_predictions, average='weighted')
best_recall = recall_score(y_test, best_predictions, average='weighted')
best_f1 = f1_score(y_test, best_predictions, average='weighted')
best_confusion = confusion_matrix(y_test, best_predictions)
best_classification_rep = classification_report(y_test, best_predictions)

# Print evaluation metrics for the best model
print("Logistic Regression Model (With Hyperparameter Tuning)")
print("Accuracy:", best_accuracy)
print("Precision:", best_precision)
print("Recall:", best_recall)
print("F1 Score:", best_f1)
print("Confusion Matrix:\n", best_confusion)
print("Classification Report:\n", best_classification_rep)
```

```
Logistic Regression Model (With Hyperparameter Tuning)
Accuracy: 0.40714285714285714
Precision: 0.3922068647678404
Recall: 0.40714285714285714
F1 Score: 0.36319090053175274
Confusion Matrix:
 [[ 0  0  0  1  0  0  0  0  0  0]
 [ 0  0  0  3  2  4  0  0  0  0]
 [ 0  0  0  0  3  1  1  0  0  0]
 [ 0  0  0 21 12 26  0  0  0  0]
 [ 0  0  0  0 12 31 10  0  0  0]
 [ 0  0  0 15  6 95 33  1  0  0]
 [ 0  0  0  1  2 35 41  1  0  0]
 [ 0  0  0  0  0 27 21  2  0  0]
 [ 0  0  0  0  0  7  1  0  0  0]
 [ 0  0  0  0  0  2  3  0  0  0]]
Classification Report:
              precision    recall  f1-score   support
```

After hyperparameter tuning: Logistic Regression





After hyperparameter tuning: Logistic Regression

Setting up Gridsearch CV:

- ParameterGrid: A dictionary (param_grid) definesthe space over which parametersshould be searched. This includes:
- C: Regularization strength, which inverselyscales with overfiting. Several values are tested from very weak tovery strong regularization.
- Solver: Different algorithms forsolving the optimization problem (newton-cg, lbfgs, liblinear). Each solver worksdifferently under various conditions and is suitable for different types of data.
- Penalty: The type of regularization applied (l2 indicates L2 regularization).

- Logistic Regression Configuration: The base model is configured with a high max_iter to ensure convergenceand set to handle multi-class classification automatically (multi_class='auto’).

- Grid Search Configuration: GridSearchCV is nitialized with the Logisic Regressionestimator, the parameter grid, and the strategy for cross-validation (cv=5 indicates 5-fold cross-validation). It aims to maximize accuracy (scoring='accuracy'), and it is settoperform computations in paralel (n_jobs=-1 uses all available CPU cores).

After hyperparameter tuning: Decision Tree

decision tree

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'max_depth': [None, 10, 20, 30, 50, 100],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10]
}
```

```
# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=param_grid, cv=5, scoring='accuracy', verbose=1, n_jobs=-1)
```

```
# Fit GridSearchCV
grid_search.fit(X_train_tfidf, y_train)
```

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
```

GridSearchCV

estimator: DecisionTreeClassifier

DecisionTreeClassifier

```
# Best parameters found
best_params = grid_search.best_params_
print("Best parameters found:", best_params)

# Best model
best_dt_model = grid_search.best_estimator_
```

```
Best parameters found: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}
```

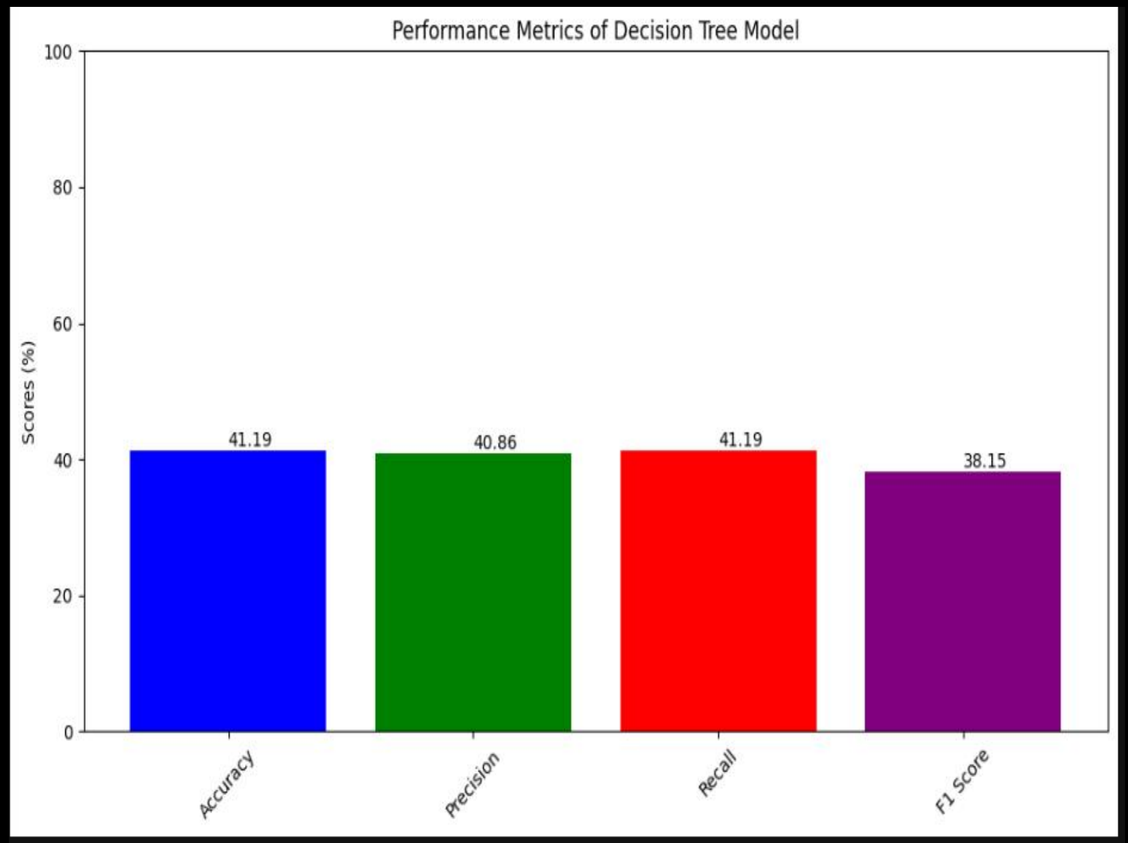
```
# Best parameters found
best_params = grid_search.best_params_
print("Best parameters found:", best_params)

# Best model
best_dt_model = grid_search.best_estimator_
```

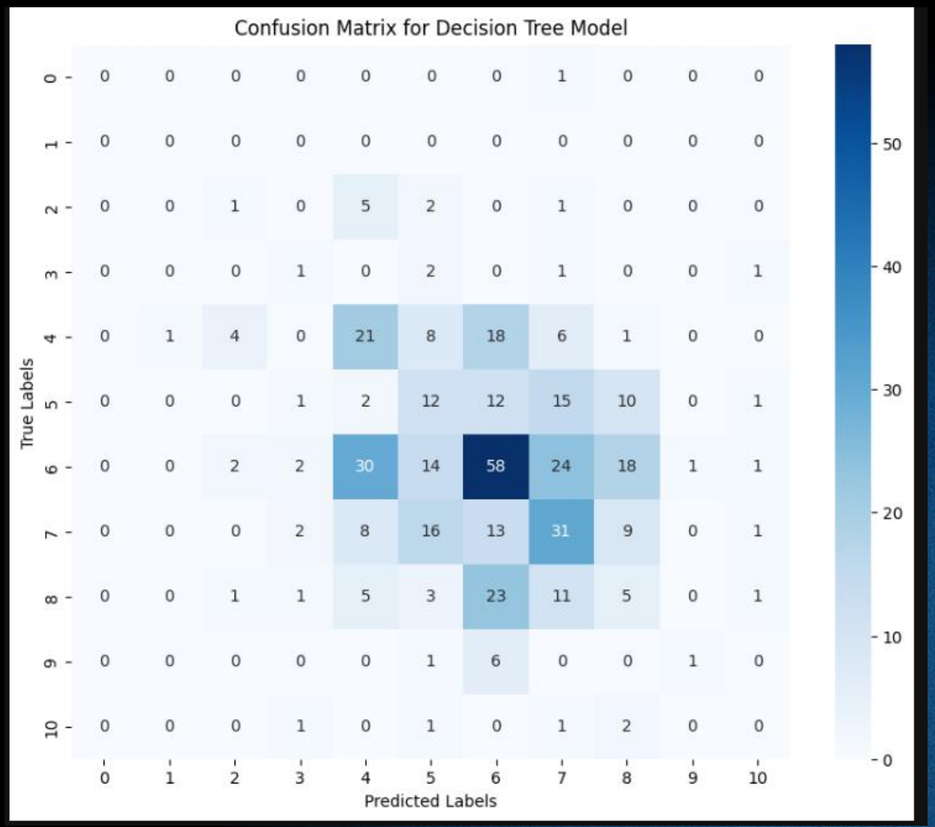
```
Best parameters found: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}
```

```
# Predictions with the best model
best_predictions = best_dt_model.predict(X_test_tfidf)

# Evaluation metrics
best_accuracy = accuracy_score(y_test, best_predictions)
best_precision = precision_score(y_test, best_predictions, average='weighted')
best_recall = recall_score(y_test, best_predictions, average='weighted')
best_f1 = f1_score(y_test, best_predictions, average='weighted')
best_confusion = confusion_matrix(y_test, best_predictions)
best_classification_rep = classification_report(y_test, best_predictions)

# Print evaluation metrics for the best model
print("Decision Tree Model (With Hyperparameter Tuning)")
print("Accuracy:", best_accuracy)
print("Precision:", best_precision)
print("Recall:", best_recall)
print("F1 Score:", best_f1)
print("Confusion Matrix:\n", best_confusion)
print("Classification Report:\n", best_classification_rep)
```

```
Decision Tree Model (With Hyperparameter Tuning)
Accuracy: 0.33095238095238094
Precision: 0.3157137307867451
Recall: 0.33095238095238094
F1 Score: 0.3106383630137287
Confusion Matrix:
 [[ 0  0  0  0  0  0  1  0  0  0]
 [ 0  0  0  4  0  1  4  0  0  0]
 [ 0  0  0  0  1  1  3  0  0  0]
 [ 0  5  0 13  8 24  8  1  0  0]
 [ 0  0  0  2 10 15 26  0  0  0]
 [ 0  0  0 17 11 67 47  8  0  0]
 [ 0  0  0  2  7 27 42  2  0  0]
 [ 0  0  0  5  6 17 15  7  0  0]
 [ 0  0  0  2  0  4  0  2  0  0]
 [ 0  0  0  0  0  1  3  1  0  0]]
```

After hyperparameter tuning: Decision Tree





After hyperparameter tuning: Decision Tree

Setting up Gridsearch CV:

- Paraeter Grid Definition: A dictionary (param_gid) is crated tspecify the rangeof values for varius hyperparameters of the DecisionTreeClassifier:
- max_depth: Contols the maxium deth (or levels) of the tree. Deepertrees can modl more comlex pattrns but are prone to overfitting.
- min_samples_split: The minimumnumber of saples required tosplit a internal node. Higher values prevet the modelrom learning overly specificpatterns, thus lowering the riskofoverfittng.
- min_samples_leaf: Theminimum number of samples a leaf node must have. This paameter further helps in controllingoverfitting.
- GridSearchCV Configuration: An instance of GridSearhCV is initializd with the Decison Tree Classifier as theetimator. It uses the definedpaameter grid to exploredifferent configrations. The seach straegy inclues:
- cv=5: 5-fold cross-validationto ensure the model's prformance is robstacross differen subsets of thedataset.
- scoring='accuracy': Optimization citerion is the accuracy of the predictions.
- vebose=1: Outputs detiled logging information during the training process.
- n_jobs=-1: Utilizes all available CPU core to parallelize the training.


Project Management

Person	Task	Description	Contribution	Issues faced
Sai Gottumukkala	Visualize and Pre-process the dataset	To clean the essay dataset using special symbols, slashes etc.	100%	Data available was less, further after cleaning and summarization, more data was lost resulting in degradation of ML models from the summary.
Sai Charan Merugu	Model analysis using Neural Network	To design a neural network model that can classify the grade	100%	Data available was less, further after cleaning and summarization, more data was lost resulting in degradation of ML models from the summary.

Person	Task	Description	Contribution	Issues faced
Swetha Guntupalli	Model analysis using Logistic Regression	To design a model that can grade the input summarized essay text.	100%	Data available was less, further after cleaning and summarization, more data was lost resulting in degradation of ML models from the summary.
Karishma Bollineni	Model analysis using decision tree	To train and implement decision tree, neural network and measure the metrics.	100%	Data available was less, further after cleaning and summarization, more data was lost resulting in degradation of ML models from the summary.

References

The Hewlett Foundation: Automated Essay Scoring
“https://www.kaggle.com/c/asap-aes/data”

Text Summarization
“https://cs.nyu.edu/~kcho/DMQA/”

Keras
“https://keras.io/keras_tuner/”

Thank you