In [ ]:
```
# Re-import libraries and redefine file paths
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import re

#  Preprocessing and Text Normalization


# Define file paths
train_file_path = r'Repositories\\Kaggle_Competitions\\Learning Agency Lab - Aut
test_file_path = r'Repositories\\Kaggle_Competitions\\Learning Agency Lab - Auto

# Load the data
train_data = pd.read_csv(train_file_path)
test_data = pd.read_csv(test_file_path)

# Display the first few rows of each dataframe and their structure
train_data.head(), train_data.columns, test_data.head(), test_data.columns
```

Out[ ]:
```
(  essay_id                                          full_text  score
 0  000d118  Many people have car where they live. The thin...      3
 1  000fe60  I am a scientist at NASA that is discussing th...      3
 2  001ab80  People always wish they had the same technolog...      4
 3  001bdc0  We all heard about Venus, the planet without a...      4
 4  002ba53  Dear, State Senator\n\nThis is a letter to arg...      3,
 Index(['essay_id', 'full_text', 'score'], dtype='object'),
   essay_id                                          full_text
 0  000d118  Many people have car where they live. The thin...
 1  000fe60  I am a scientist at NASA that is discussing th...
 2  001ab80  People always wish they had the same technolog...,
 Index(['essay_id', 'full_text'], dtype='object'))
```

Preprocessing and Text Normalization

- Cleaning Text: Remove or normalize text artifacts like punctuation, capitalization, and special characters that might not contribute to essay scoring.
- Tokenization and Lemmatization: Break down text into tokens (words or phrases) and reduce them to their base or dictionary form.
- Stopword Removal: Consider the impact of removing common words that may not contribute to the overall meaning of the essay.

In [ ]:
```
# Setting up NLTK with local resources
nltk.data.path.append('Repositories\\Kaggle_Competitions\\Learning Agency Lab -

# Load NLTK resources necessary for the tasks
nltk.download('punkt', download_dir='Repositories\\Kaggle_Competitions\\Learning
nltk.download('stopwords', download_dir='Repositories\\Kaggle_Competitions\\Lear
nltk.download('wordnet', download_dir='Repositories\\Kaggle_Competitions\\Learni
```

```
[nltk_data] Downloading package punkt to C:\Users\nickr\OneDrive\Υπολο
[nltk_data]     γιστής\Repositories\Kaggle_Competitions\Learning
[nltk_data]     Agency Lab - Automated Essay Scoring 2.0\...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to C:\Users\nickr\OneDrive\Υ
[nltk_data]     πολογιστής\Repositories\Kaggle_Competitions\Learning
[nltk_data]     Agency Lab - Automated Essay Scoring 2.0\...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\nickr\OneDrive\Υπο
[nltk_data]     λογιστής\Repositories\Kaggle_Competitions\Learning
[nltk_data]     Agency Lab - Automated Essay Scoring 2.0\...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[ ]: True

In [ ]:

```
# Initialize the WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# Function to preprocess text
def preprocess_text(text):
    # Convert text to lower case
    text = text.lower()
    # Remove non-alphabetic characters and extra spaces
    text = re.sub('[^a-z\s]', ' ', text)
    text = re.sub(' +', ' ', text).strip()
    # Tokenize text
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    # Lemmatize words
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    # Join tokens back to string
    text = ' '.join(tokens)
    return text

# Apply preprocessing to both train and test data
train_data['clean_text'] = train_data['full_text'].apply(preprocess_text)
test_data['clean_text'] = test_data['full_text'].apply(preprocess_text)

# Display first few rows to verify preprocessing
train_data[['full_text', 'clean_text']].head(), test_data[['full_text', 'clean_t
```

Out[ ]:
```
(                                            full_text  \
 0  Many people have car where they live. The thin...
 1  I am a scientist at NASA that is discussing th...
 2  People always wish they had the same technolog...
 3  We all heard about Venus, the planet without a...
 4  Dear, State Senator\n\nThis is a letter to arg...

                                            clean_text
 0  many people car live thing know use car alot t...
 1  scientist nasa discussing face mar explaining ...
 2  people always wish technology seen movie best ...
 3  heard venus planet without almost oxygen earth...
 4  dear state senator letter argue favor keeping ...  ,
                                             full_text  \
 0  Many people have car where they live. The thin...
 1  I am a scientist at NASA that is discussing th...
 2  People always wish they had the same technolog...

                                            clean_text
 0  many people car live thing know use car alot t...
 1  scientist nasa discussing face mar explaining ...
 2  people always wish technology seen movie best ...  )
```

2.Feature Engineering

- Linguistic Features: Extract features that represent the quality of writing, such as sentence complexity, vocabulary richness, grammar correctness, and coherence. Tools like the Natural Language Toolkit (NLTK) or spaCy can be helpful.
- Text Embeddings: Use embeddings like Word2Vec, GloVe, or fastText to capture semantic relationships between words. Sentence and paragraph embeddings (e.g., from BERT or Sentence-BERT) can capture contextual nuances.
- Syntactic Features: Parse trees and dependency graphs can help understand the syntactic structures of sentences, potentially indicating more complex writing abilities.

In [ ]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

# Function to calculate linguistic features
def linguistic_features(text):
    sentences = text.split('.')
    sentence_lengths = [len(sentence.split()) for sentence in sentences if sente

    # Average sentence length
    avg_sentence_length = np.mean(sentence_lengths) if sentence_lengths else 0

    # Vocabulary richness: Type-Token Ratio (TTR)
    words = text.split()
    unique_words = set(words)
    ttr = len(unique_words) / len(words) if words else 0

    return avg_sentence_length, ttr

# Apply linguistic features calculation
train_data['avg_sentence_length'], train_data['ttr'] = zip(*train_data['clean_te
test_data['avg_sentence_length'], test_data['ttr'] = zip(*test_data['clean_text'
```

```
# Initialize a TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=100)  # Limit number of features

# Fit and transform the 'clean_text' column to create TF-IDF features
tfidf_train = tfidf_vectorizer.fit_transform(train_data['clean_text'])
tfidf_test = tfidf_vectorizer.transform(test_data['clean_text'])

# Example: Convert first 5 TF-IDF features of train data to dense format and dis
tfidf_train_dense_example = tfidf_train.todense()[:5]

tfidf_train_dense_example
```

Out[ ]:
```
matrix([[0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.04548181,
         0.79017261, 0.04975925, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.04778838, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.0321821 , 0.1117614 ,
         0.        , 0.19593373, 0.03828935, 0.        , 0.08631919,
         0.03679849, 0.        , 0.        , 0.078892  , 0.0273742 ,
         0.        , 0.        , 0.        , 0.03337794, 0.06588664,
         0.        , 0.        , 0.        , 0.04189591, 0.        ,
         0.11405601, 0.04082721, 0.03086056, 0.        , 0.13427865,
         0.32417327, 0.        , 0.        , 0.1876218 , 0.        ,
         0.        , 0.        , 0.        , 0.04384533, 0.04065578,
         0.        , 0.15026438, 0.        , 0.        , 0.        ,
         0.07776394, 0.        , 0.05145062, 0.        , 0.11402972,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.13661017, 0.10336992, 0.        ,
         0.17038731, 0.        , 0.        , 0.        , 0.03322437,
         0.        , 0.        , 0.04177855, 0.        , 0.09694338],
        [0.        , 0.        , 0.        , 0.12031148, 0.        ,
         0.06052403, 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.08259009, 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.13234917, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.38101442,
         0.        , 0.        , 0.06502235, 0.0445639 , 0.        ,
         0.        , 0.        , 0.        , 0.10085184, 0.        ,
         0.10191284, 0.54652212, 0.        , 0.10924507, 0.1516248 ,
         0.13987506, 0.        , 0.062744  , 0.04621983, 0.091236  ,
         0.39450627, 0.        , 0.        , 0.        , 0.36208903,
         0.        , 0.        , 0.17093564, 0.        , 0.03718826,
         0.        , 0.        , 0.08350221, 0.        , 0.06603448,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.20270679, 0.07770461, 0.        ,
         0.        , 0.        , 0.        , 0.06131636, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.07013054, 0.09458492, 0.09542712, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.06056744, 0.        , 0.10863886, 0.        ],
        [0.03389527, 0.17204791, 0.        , 0.04384889, 0.        ,
         0.06617602, 0.        , 0.        , 0.        , 0.        ,
         0.69263231, 0.        , 0.        , 0.        , 0.        ,
         0.06772703, 0.        , 0.        , 0.        , 0.11619127,
         0.03865233, 0.24130051, 0.17688265, 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.14446337, 0.        ,
         0.        , 0.03987428, 0.        , 0.12181368, 0.05640431,
         0.        , 0.        , 0.        , 0.02756745, 0.16336503,
         0.        , 0.        , 0.04107377, 0.        , 0.04144604,
         0.        , 0.03662262, 0.        , 0.02526802, 0.14963401,
         0.        , 0.16234906, 0.        , 0.        , 0.        ,
         0.05756236, 0.30907322, 0.04672459, 0.        , 0.06099159,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.04743942, 0.        , 0.07807622, 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.11684862, 0.03352117, 0.0575491 ,
         0.03295229, 0.        , 0.06314252, 0.02752881, 0.        ,
         0.32421573, 0.        , 0.        , 0.02608463, 0.0238461 ,
         0.        , 0.        , 0.        , 0.06239279, 0.05030353,
         0.03407168, 0.03311174, 0.        , 0.        , 0.        ],
```

```
       [0.        , 0.        , 0.        , 0.0914367 , 0.        ,
        0.04599825, 0.05034357, 0.32843039, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.11305302, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.10058534, 0.        ,
        0.        , 0.        , 0.        , 0.03347165, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.11497108, 0.04542132,
        0.        , 0.        , 0.        , 0.04151312, 0.05761741,
        0.        , 0.        , 0.        , 0.1053812 , 0.03466967,
        0.        , 0.        , 0.        , 0.        , 0.11007504,
        0.        , 0.0429667 , 0.        , 0.        , 0.02826307,
        0.05686019, 0.        , 0.        , 0.        , 0.45167574,
        0.        , 0.        , 0.        , 0.04614299, 0.0427863 ,
        0.        , 0.        , 0.        , 0.17716646, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.04580957, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.1065984 , 0.03594226, 0.        , 0.        ,
        0.        , 0.73031196, 0.        , 0.        , 0.0699309 ,
        0.        , 0.        , 0.        , 0.11008736, 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.03359203,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.56157192, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.099039  ,
        0.1009494 , 0.56797115, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.02474935, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.12517673, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.02533476,
        0.        , 0.03298508, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.02373303, 0.        , 0.02065316,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.19276898, 0.13886939, 0.        , 0.        , 0.12506386,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.17538706,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.46804938, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ]])
```

3. Model Selection and Ensemble Methods

- Advanced NLP Models: Utilize state-of-the-art language models like BERT, GPT, or their variants (RoBERTa, DistilBERT, etc.) fine-tuned on the essay dataset.
- Ensemble Methods: Combine predictions from multiple models to improve accuracy. Techniques like bagging, boosting, or stacking can be particularly effective, especially when combining models that capture different aspects of writing quality.
- Custom Scoring Layers: For neural networks, consider designing custom layers or loss functions that directly optimize for the competition's evaluation metric (Quadratic Weighted Kappa).

In [ ]:
```
import torch
import transformers
print("Torch version:", torch.__version__)
print("Transformers version:", transformers.__version__)
```

```
Torch version: 2.2.2
Transformers version: 4.39.3
```

In [ ]:
```
from transformers import BertModel, BertTokenizer
import torch
import torch.nn as nn

# Specify the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load pre-trained BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model_bert = BertModel.from_pretrained('bert-base-uncased').to(device)  # Move m

class BERTRegressor(nn.Module):
    def __init__(self):
        super(BERTRegressor, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased').to(device)  #
        self.regressor = nn.Linear(768, 1)  # Assuming the output of BERT is 768

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids, attention_mask=attention_mask, return_dic
        return self.regressor(outputs.pooler_output)  # Using the pooled output

# Instantiate the model
regressor_model = BERTRegressor().to(device)  # Move the entire model to the rig

# Example forward pass
inputs = tokenizer("Example text input for BERT", return_tensors="pt")
inputs = {key: value.to(device) for key, value in inputs.items()}  # Move input

score = regressor_model(inputs['input_ids'], inputs['attention_mask'])
print(score)
```

```
tensor([[0.3053]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

In [ ]:
```
print("Model device:", next(regressor_model.parameters()).device)
print("Input device:", inputs['input_ids'].device)
```

```
Model device: cuda:0
Input device: cuda:0
```

In [ ]:
```
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_wi
import torch
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler, Tensc
import torch.nn as nn
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
import numpy as np

# Parameters
PRE_TRAINED_MODEL_NAME = 'bert-base-uncased'
BATCH_SIZE = 16
EPOCHS = 3
MAX_LEN = 256  # Maximum length of tokens

# GPU or CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Tokenizer
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
# Data Preparation Functions
def create_data_loader(df, tokenizer, max_len, batch_size):
    """Converts text data into a torch DataLoader."""
    token_ids = []
    attention_masks = []

    # Ensure the column names here match those in your DataFrame
    for text in df['full_text']:  # Assuming 'full_text' is the correct column n
        encoding = tokenizer.encode_plus(
            text,
            max_length=max_len,
            add_special_tokens=True,
            return_token_type_ids=False,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt',
            truncation=True
        )

        token_ids.append(encoding['input_ids'])
        attention_masks.append(encoding['attention_mask'])

    token_ids = torch.cat(token_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)

    # Correct the column name for targets if different
    targets = torch.tensor(df['score'].values)  # Use the correct column name fo

    dataset = TensorDataset(token_ids, attention_masks, targets)
    sampler = RandomSampler(dataset)
    loader = DataLoader(dataset, sampler=sampler, batch_size=batch_size)

    return loader


# Load data
train_loader = create_data_loader(train_data, tokenizer, MAX_LEN, BATCH_SIZE)
test_loader = create_data_loader(test_data, tokenizer, MAX_LEN, BATCH_SIZE)

# BERT Model Setup for Regression
class BERTRegressor(nn.Module):
    def __init__(self):
        super(BERTRegressor, self).__init__()
        self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
        self.drop = nn.Dropout(p=0.3)
        self.out = nn.Linear(self.bert.config.hidden_size, 1)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask,
        # Use outputs.pooler_output if you're using a pooled output for regressi
        # It represents the entire sequence context pooled into a single 768-len
        pooled_output = outputs.pooler_output
        output = self.drop(pooled_output)  # Apply dropout to the pooled output
        return self.out(output)  # Apply the linear layer and return

model = BERTRegressor()
model = model.to(device)

# Loss and Optimizer
```

```
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_loader) * EPOCHS
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=total_steps
)
loss_fn = nn.MSELoss().to(device)

# Training Loop
def train_epoch(model, data_loader, loss_fn, optimizer, device, scheduler, n_exa
    model = model.train()
    losses = []

    for d in tqdm(data_loader):
        input_ids = d[0].to(device)
        attention_mask = d[1].to(device)
        targets = d[2].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        outputs = outputs.squeeze()  # Ensure the output is squeezed to match ta

        loss = loss_fn(outputs, targets.float())

        losses.append(loss.item())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        scheduler.step()

    return np.mean(losses)


# Run Training
for epoch in range(EPOCHS):
    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-' * 10)

    train_loss = train_epoch(
        model,
        train_loader,
        loss_fn,
        optimizer,
        device,
        scheduler,
        len(train_data)
    )

    print(f'Train loss {train_loss}')
```

In [ ]:

```
print(train_data.columns)  # This should include both 'full_text' and 'score'
train_loader = create_data_loader(train_data, tokenizer, MAX_LEN, BATCH_SIZE)
test_loader = create_data_loader(test_data, tokenizer, MAX_LEN, BATCH_SIZE)
```

```
Index(['essay_id', 'full_text', 'score', 'clean_text', 'avg_sentence_length',
       'ttr'],
      dtype='object')
```