Download data from Kaggle

```
!pip install -q kaggle
```

```
from google.colab import files
# Create a new API token under "Account" in the kaggle webpage and download the json file
# Upload the file by clicking on the browse
files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to e

```
Saving kaggle.json to kaggle.json
```

{'kaggle.json': b'{"username":"aditivijaykulkarni","key":"8b1a498f4da2b10723fa3ea7cf8d8ae3"}'}

```
! mkdir ~/.kaggle
```

```
! cp kaggle.json ~/.kaggle/
```

```
!kaggle competitions download -c commonlit-evaluate-student-summaries
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle
Downloading commonlit-evaluate-student-summaries.zip to /content
 95% 1.00M/1.05M [00:00<00:00, 2.00MB/s]
100% 1.05M/1.05M [00:00<00:00, 2.09MB/s]
```

Extract data and install packages (regardless of data acquisition method)

```
!unzip commonlit-evaluate-student-summaries.zip
```

```
Archive:  commonlit-evaluate-student-summaries.zip
  inflating: prompts_test.csv
  inflating: prompts_train.csv
  inflating: sample_submission.csv
  inflating: summaries_test.csv
  inflating: summaries_train.csv
```

```
### TODO: Install required packages
### Student's code here
!pip install pandas
!pip install scikit-learn
!pip install numpy
!pip install matplotlib
!pip install seaborn
!pip install nltk
!pip install pyphen
!pip install pyspellchecker
!pip install language_tool_python
### END
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.(
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlit
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.23.5)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.10/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.10/dist-packages (from seaborn) (3.7.1)
```

```
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->sea
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1->se
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=3.1-
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib!=3.6.1,>=
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25->seaborn) (2023
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
Collecting pyphen
  Downloading pyphen-0.14.0-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 12.8 MB/s eta 0:00:00
Installing collected packages: pyphen
Successfully installed pyphen-0.14.0
Collecting pyspellchecker
  Downloading pyspellchecker-0.7.2-py3-none-any.whl (3.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 17.8 MB/s eta 0:00:00
Installing collected packages: pyspellchecker
Successfully installed pyspellchecker-0.7.2
Collecting language_tool_python
  Downloading language_tool_python-2.7.1-py3-none-any.whl (34 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from language_tool_python) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from language_tool_python) (4.66.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->langua
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->language_tool_pyth
```

Section 1: Library and Data Imports (Q1, 5 points)

- Import your libraries and join the data from both `summaries_train.csv` and `prompts_train.csv` into a single dataframe with the same structure as `use_cols`. Print the head of the dataframe. **Do not modify `use_cols`.**

```python
### TODO: Load required packages
### Student's code here
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
nltk.download('popular')
import pyphen
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
nltk.download('stopwords')
from sklearn.feature_extraction.text import CountVectorizer
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
from spellchecker import SpellChecker
from language_tool_python import LanguageTool
from scipy import stats
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
###
```

```
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package gazetteers to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gazetteers.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gutenberg.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/shakespeare.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/stopwords.zip.
[nltk_data]    | Downloading package treebank to /root/nltk_data...
```

```
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/twitter_samples.zip.
[nltk_data]    | Downloading package omw to /root/nltk_data...
[nltk_data]    | Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet to /root/nltk_data...
[nltk_data]    | Downloading package wordnet2021 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet31 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/wordnet_ic.zip.
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/words.zip.
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data]    | Downloading package punkt to /root/nltk_data...
[nltk_data]    |   Unzipping tokenizers/punkt.zip.
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    |
[nltk_data]  Done downloading collection popular
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
use_cols = ["student_id",
            "prompt_id",
            "text",
            "content",
            "wording",
            "prompt_question",
            "prompt_title",
            "prompt_text"
           ]
dtypes = {
        'student_id':                                   'string',
        'prompt_id':                                    'string',
        'text':                                         'string',
        'content':                                      'Float64',
        'wording':                                      'Float64',
        'prompt_question':                              'string',
        'prompt_title':                                 'string',
        'prompt_text':                                  'string',
        }
```

```
# reading two csv files

data1 = pd.read_csv('summaries_train.csv')
data2 = pd.read_csv('prompts_train.csv')

# using merge function
mergedDF = pd.merge(data1, data2,
                   on='prompt_id')

mergedDF = mergedDF[use_cols]
mergedDF.head()
```

	student_id	prompt_id	text	content	wording	prompt_question	prompt_title
0	000e8c3c7ddb	814d6b	The third wave was an experimentto see how peo...	0.205683	0.380538	Summarize how the Third Wave developed over su...	The Third Wave
1	0070c9e7af47	814d6b	The Third Wave developed rapidly because the ...	3.272894	3.219757	Summarize how the Third Wave developed over su...	The Third Wave
2	0095993991fe	814d6b	The third wave only started as an experiment w...	0.205683	0.380538	Summarize how the Third Wave developed over su...	The Third Wave
3	00c20c6ddd23	814d6b	The experimen was orginally about how even whe...	0.567975	0.969062	Summarize how the Third Wave developed over su...	The Third Wave
4	00d40ad10dc9	814d6b	The third wave developed so quickly due to the...	-0.910596	-0.081769	Summarize how the Third Wave developed over su...	The Third Wave

```
from pandas.core.reshape.merge import string
# convert column data types to string

mergedDF[['student_id', 'prompt_id', 'text', 'prompt_question', 'prompt_title', 'prompt_text']] = mergedDF[['student_id', 'prc

mergedDF.dtypes
```

```
student_id           string
prompt_id            string
text                 string
content             float64
wording             float64
prompt_question      string
prompt_title         string
prompt_text          string
dtype: object
```

Section 2: Features (Q2 and Q3, 25 points total)

```
# Construct a table of five features (really 7) from the text for each instance: (10 points)
# Number of words in student response (text) and prompt (prompt_text)
# Number of distinct words in student response (text) and prompt (prompt_text)
# Number of words common to student response (text) and prompt (prompt_text)
# Number of words common to student response (text) and prompt_question
# Number of words common to student response (text) and prompt_title

d = {
    'student_id': mergedDF["student_id"],
    'no_of_words_in_text': mergedDF["text"].str.split().apply(len),
    'no_of_words_in_prompt_text': mergedDF["prompt_text"].str.split().apply(len),
    'no_of_distinct_words_in_text': mergedDF["text"].str.lower().str.split().apply(set).apply(len),
    'no_of_distinct_words_in_prompt_text': mergedDF["prompt_text"].str.lower().str.split().apply(set).apply(len),
    'no_of_words_in_text_and_prompt_text': mergedDF["text"].str.split().apply(len) + mergedDF["prompt_text"].str.split().apply
    'no_of_distinct_words_in_text_and_prompt_text': mergedDF['text'].str.lower().str.split().apply(set).apply(len) + mergedDF[
    'no_of_words_common_to_text_and_prompt_text': mergedDF.apply(lambda row: len(set(word.lower() for word in row["text"].spli
    'no_of_words_common_to_text_and_prompt_question': mergedDF.apply(lambda row: len(set(word.lower() for word in row["text"].
    'no_of_words_common_to_text_and_prompt_title': mergedDF.apply(lambda row: len(set(word.lower() for word in row["text"].spl
}

df = pd.DataFrame(data=d)

df.head()
```

	student_id	no_of_words_in_text	no_of_words_in_prompt_text	no_of_distinct_words_in_text	no_of_distinct_words_in_pr
0	000e8c3c7ddb	61	596	49	
1	0070c9e7af47	203	596	134	
2	0095993991fe	60	596	48	
3	00c20c6ddd23	76	596	57	
4	00d40ad10dc9	27	596	24	

```
# Now fortify this list with at least five other numerical features.
# Consider readability indices, counts of words from particular classes (e.g character length, part of speech, popularity).
# Use your imagination as to what might be helpful for identifying well written summaries of texts.

# total sentences
df['total_sentences_in_text'] = mergedDF['text'].apply(lambda text: len(nltk.sent_tokenize(text)))

# total syllables
dictionary = pyphen.Pyphen(lang='en')
df['total_syllables_in_text'] = mergedDF['text'].apply(lambda text: sum(len(dictionary.inserted(word).split('-')) for word in

# readability index Flesch–Kincaid grade level
df['readability_index'] =  206.835 - 1.015 * (df['no_of_words_in_text']/df['total_sentences_in_text']) - 84.6 * (df['total_syl

# Type-Token Ratio (TTR)/Unique words
df['type_token_ratio'] = df['no_of_distinct_words_in_text'] / df['no_of_words_in_text']

# Number of words in text without punctuation and stop words
df['num_words_no_punctuation_stopwords'] = mergedDF['text'].apply(lambda x: len([word for word in nltk.word_tokenize(x) if wor

# POS Tagging
df['pos_tags'] = mergedDF['text'].apply(lambda x: [tag for word, tag in nltk.pos_tag(nltk.word_tokenize(x))])

# Number of misspelled words
spell = SpellChecker()
count_misspelled_words = lambda text: len(spell.unknown(text.split()))
df['misspelled_word_count'] = mergedDF['text'].apply(count_misspelled_words)

# Number of grammatical errors
tool = LanguageTool('en-US')
```

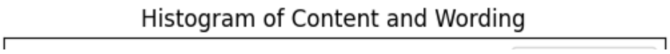
```
df['grammar_error_count'] = mergedDF['text'].apply(lambda text: len(tool.check(text)))

df.head()
```

```
Downloading LanguageTool 5.7: 100%|██████████| 225M/225M [00:04<00:00, 49.4MB/s]
INFO:language_tool_python.download_lt:Unzipping /tmp/tmppeptgz1b.zip to /root/.cache/language_tool_python.
INFO:language_tool_python.download_lt:Downloaded https://www.languagetool.org/download/LanguageTool-5.7.zip to /root/.cac
```

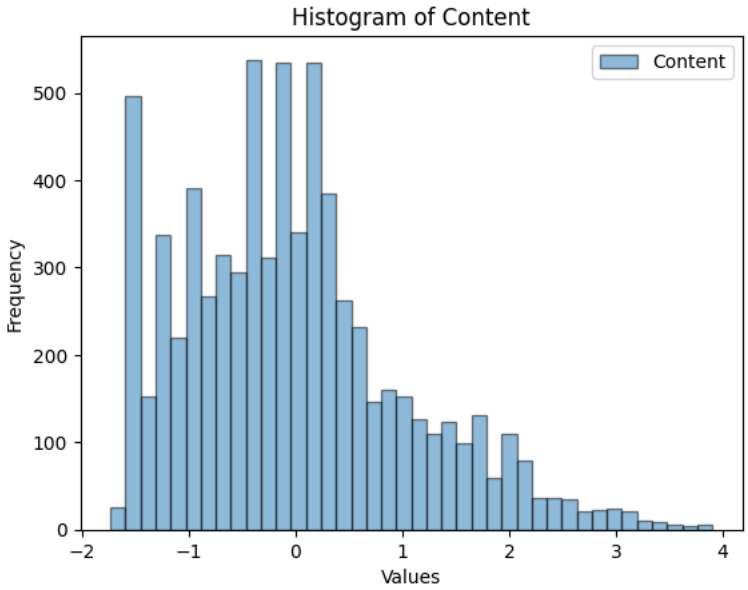
	student_id	no_of_words_in_text	no_of_words_in_prompt_text	no_of_distinct_words_in_text	no_of_distinct_words_in_pr
0	000e8c3c7ddb	61	596	49	
1	0070c9e7af47	203	596	134	
2	0095993991fe	60	596	48	
3	00c20c6ddd23	76	596	57	
4	00d40ad10dc9	27	596	24	

Section 3: Content and Wording (Q4, 10 points)

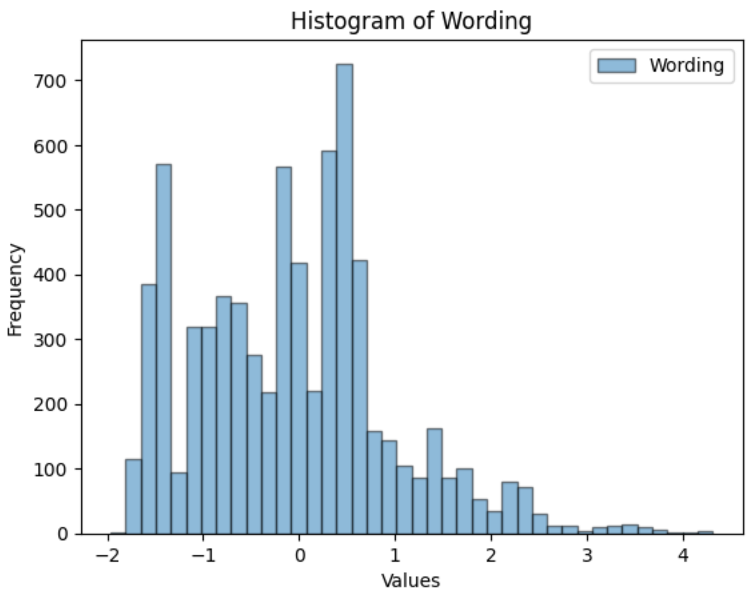
```
# Look at the distributions of scores for content and wording, as histograms and scatterplots?
# What is the range of values here? How well correlated are they?
# Do the shapes of these distributions differ for the different prompts?  (10 points)

# histogram of content and wording
plt.hist(mergedDF['content'], bins=40, alpha=0.5, label='Content', edgecolor='black')
plt.hist(mergedDF['wording'], bins=40, alpha=0.5, label='Wording', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Content and Wording')
plt.show()
```



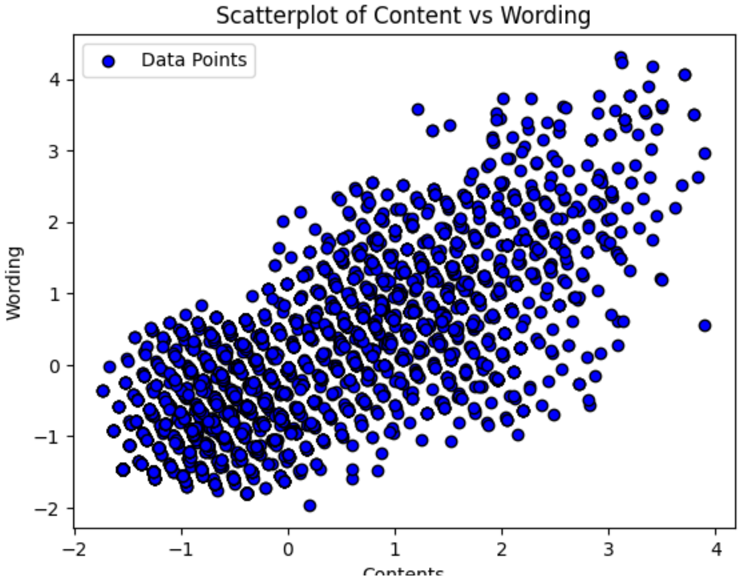

```
# histogram of content

plt.hist(mergedDF['content'], bins=40, alpha=0.5, label='Content', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Content')
plt.show()
```




```
# histogram of wording

plt.hist(mergedDF['wording'], bins=40, alpha=0.5, label='Wording', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Wording')
plt.show()
```




```
# scatterplot of content vs wording
plt.scatter(mergedDF['content'], mergedDF['wording'], label='Data Points', color='blue', marker='o', edgecolor='black')

plt.xlabel('Contents')
plt.ylabel('Wording')
plt.title('Scatterplot of Content vs Wording')
plt.legend()
plt.show()
```




```
# Find the range of 'Content'
content_range = mergedDF['content'].max() - mergedDF['content'].min()

print("Range of 'Content':", content_range)

wording_range = mergedDF['wording'].max() - mergedDF['wording'].min()

print("Range of 'Wording':", wording_range)
```

```
Range of 'Content': 5.630185556899
Range of 'Wording': 6.27330694515344
```

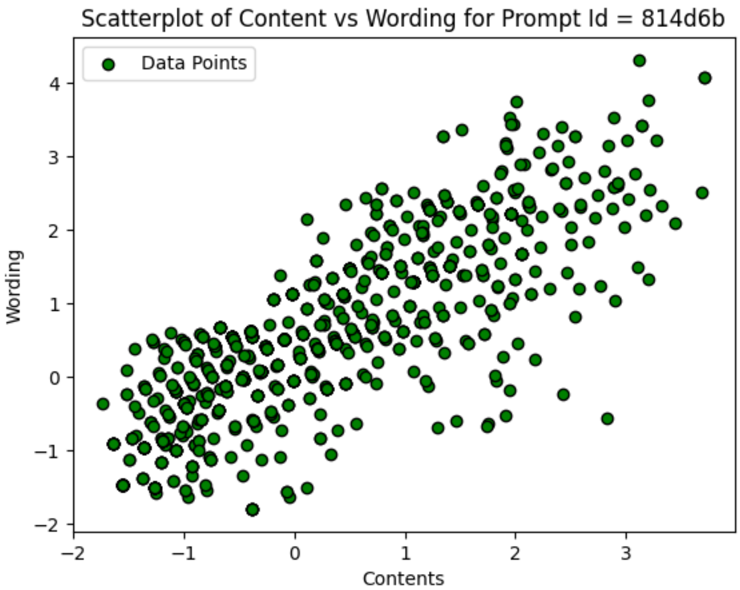
```
# find the correlation of content and wording

correlation = mergedDF['content'].corr(mergedDF['wording'])
print(f'Correlation between cotent and wording - {correlation}')

# Content and wording seem to have a strong positive correlation which in turn translates to that if content is increasing, wc
```

```
Correlation between cotent and wording - 0.7513804859701969
```

```
unique_prompt_ids = mergedDF['prompt_id'].unique()
print(unique_prompt_ids)

# histogram for prompt id = 814d6b
plt.hist(mergedDF[mergedDF['prompt_id'] == '814d6b']['content'], bins=40, alpha=0.5, label='Content', edgecolor='black')
plt.hist(mergedDF[mergedDF['prompt_id'] == '814d6b']['wording'], bins=40, alpha=0.5, label='Wording', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Content and Wording for Prompt Id = 814d6b')
plt.show()
```

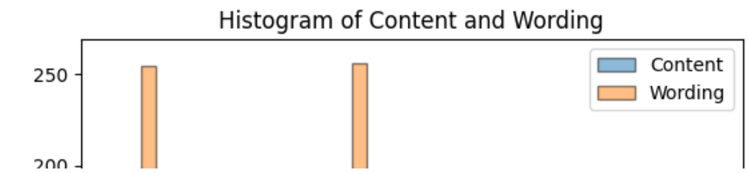
```
<StringArray>
['814d6b', 'ebad26', '3b9047', '39c16e']
Length: 4, dtype: string
```

```
# scatterplot for prompt id = 814d6b
plt.scatter(mergedDF[mergedDF['prompt_id'] == '814d6b']['content'], mergedDF[mergedDF['prompt_id'] == '814d6b']['wording'], la

plt.xlabel('Contents')
plt.ylabel('Wording')
plt.title('Scatterplot of Content vs Wording for Prompt Id = 814d6b')
plt.legend()
plt.show()

# Find the range for prompt_id = 814d6b
content_range = mergedDF[mergedDF['prompt_id'] == '814d6b']['content'].max() - mergedDF[mergedDF['prompt_id'] == '814d6b']['cc

print("Range of 'Content':", content_range)

wording_range = mergedDF[mergedDF['prompt_id'] == '814d6b']['wording'].max() - mergedDF[mergedDF['prompt_id'] == '814d6b']['wc

print("Range of 'Wording':", wording_range)

# find the correlation of content and wording for prompt_id = 814d6b

correlation = mergedDF[mergedDF['prompt_id'] == '814d6b']['content'].corr(mergedDF[mergedDF['prompt_id'] == '814d6b']['wording
print(f'Correlation between content and wording for prompt id = 814d6b - {correlation}')
```

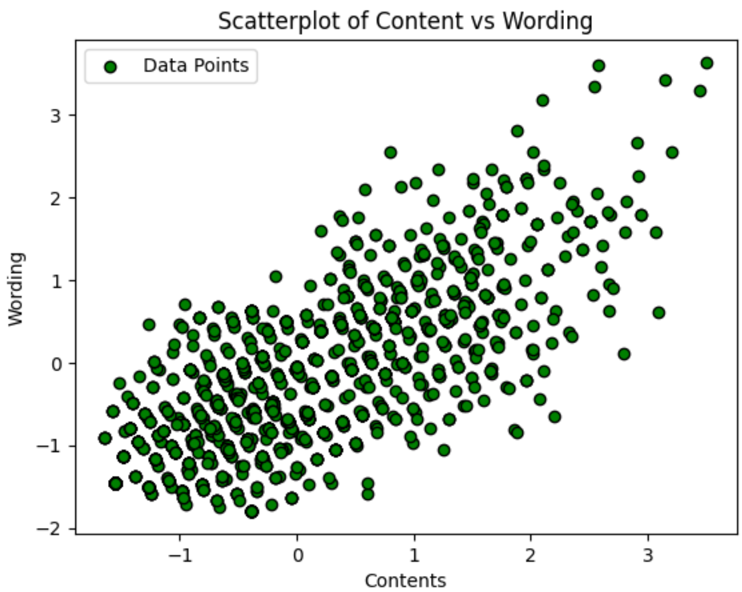



```
Range of 'Content': 5.4412334055625005
Range of 'Wording': 6.10618395484142
Correlation between content and wording for prompt id = 814d6b - 0.8137912322007284
```

```
# histogram for prompt id = ebad26

plt.hist(mergedDF[mergedDF['prompt_id'] == 'ebad26']['content'], bins=40, alpha=0.5, label='Content', edgecolor='black')
plt.hist(mergedDF[mergedDF['prompt_id'] == 'ebad26']['wording'], bins=40, alpha=0.5, label='Wording', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Content and Wording')
plt.show()
```



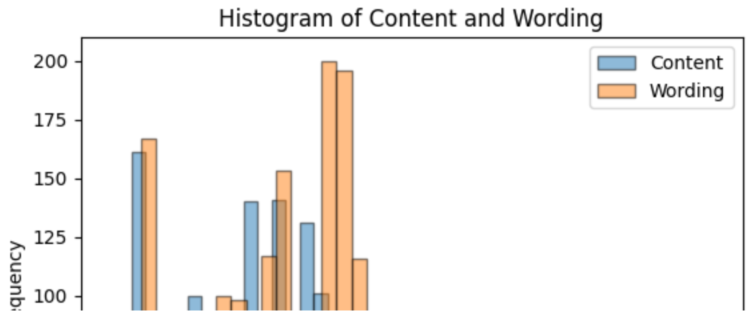

```
# scatterplot for prompt id = ebad26
plt.scatter(mergedDF[mergedDF['prompt_id'] == 'ebad26']['content'], mergedDF[mergedDF['prompt_id'] == 'ebad26']['wording'], la

plt.xlabel('Contents')
plt.ylabel('Wording')
plt.title('Scatterplot of Content vs Wording')
plt.legend()
plt.show()
```



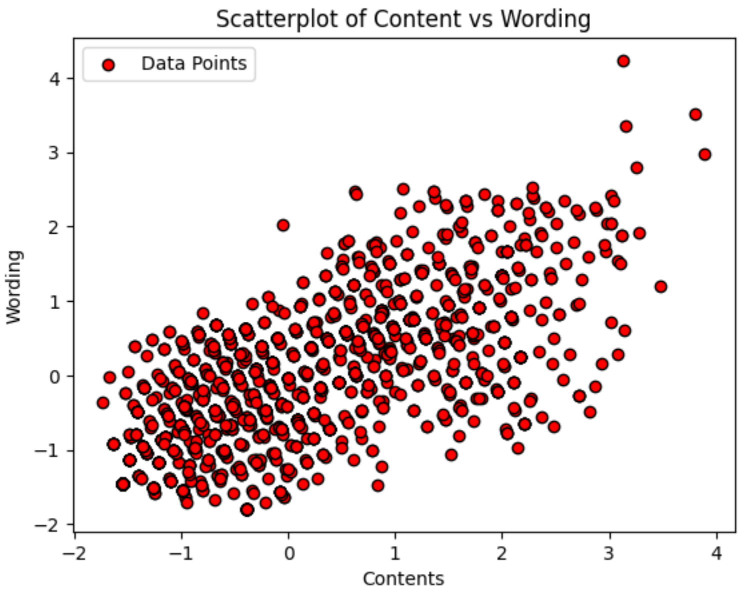

```
# Find the range for prompt_id = ebad26
content_range = mergedDF[mergedDF['prompt_id'] == 'ebad26']['content'].max() - mergedDF[mergedDF['prompt_id'] == 'ebad26']['cc

print("Range of 'Content':", content_range)

wording_range = mergedDF[mergedDF['prompt_id'] == 'ebad26']['wording'].max() - mergedDF[mergedDF['prompt_id'] == 'ebad26']['wc

print("Range of 'Wording':", wording_range)

# find the correlation of content and wording for prompt_id = ebad26

correlation = mergedDF[mergedDF['prompt_id'] == 'ebad26']['content'].corr(mergedDF[mergedDF['prompt_id'] == 'ebad26']['wording
print(f'Correlation between content and wording - {correlation}')
```

```
Range of 'Content': 5.14173706736242
Range of 'Wording': 5.43361713252189
Correlation between content and wording - 0.7534131842752712
```

```
# histogram for prompt id = 3b9047

plt.hist(mergedDF[mergedDF['prompt_id'] == '3b9047']['content'], bins=40, alpha=0.5, label='Content', edgecolor='black')
plt.hist(mergedDF[mergedDF['prompt_id'] == '3b9047']['wording'], bins=40, alpha=0.5, label='Wording', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Content and Wording')
plt.show()
```



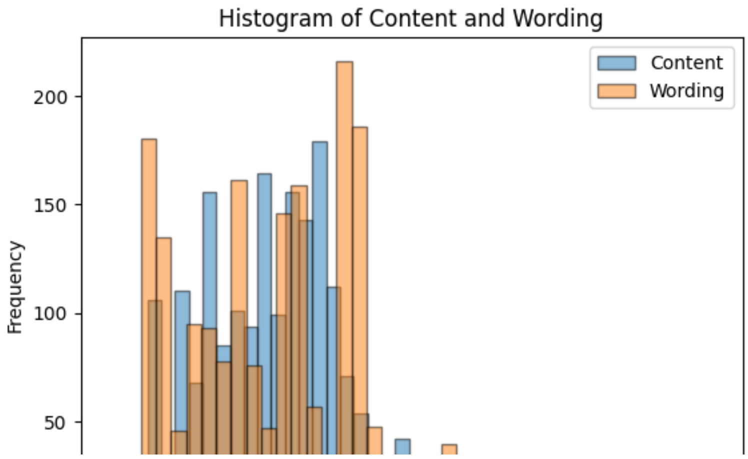

```
# scatterplot for prompt id = 3b9047

plt.scatter(mergedDF[mergedDF['prompt_id'] == '3b9047']['content'], mergedDF[mergedDF['prompt_id'] == '3b9047']['wording'], la
plt.xlabel('Contents')
plt.ylabel('Wording')
plt.title('Scatterplot of Content vs Wording')
plt.legend()
plt.show()
```



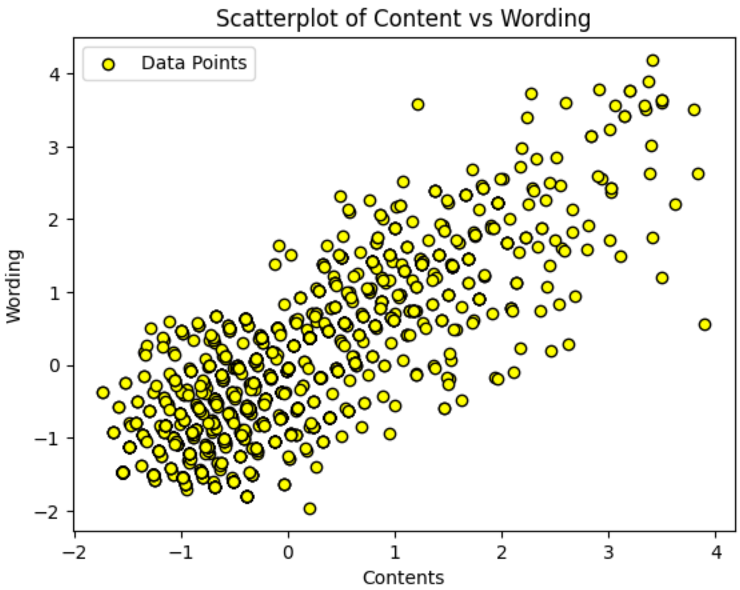

```
# Find the range for prompt_id = 3b9047
content_range = mergedDF[mergedDF['prompt_id'] == '3b9047']['content'].max() - mergedDF[mergedDF['prompt_id'] == '3b9047']['cc

print("Range of 'Content':", content_range)

wording_range = mergedDF[mergedDF['prompt_id'] == '3b9047']['wording'].max() - mergedDF[mergedDF['prompt_id'] == '3b9047']['wc

print("Range of 'Wording':", wording_range)

# find the correlation of content and wording for prompt_id = 3b9047

correlation = mergedDF[mergedDF['prompt_id'] == '3b9047']['content'].corr(mergedDF[mergedDF['prompt_id'] == '3b9047']['wording
print(f'Correlation between content and wording - {correlation}')
```

```
Range of 'Content': 5.62392964131342
Range of 'Wording': 6.02671635569877
Correlation between content and wording - 0.687493601347762
```

```
# histogram for prompt id = 39c16e

plt.hist(mergedDF[mergedDF['prompt_id'] == '39c16e']['content'], bins=40, alpha=0.5, label='Content', edgecolor='black')
plt.hist(mergedDF[mergedDF['prompt_id'] == '39c16e']['wording'], bins=40, alpha=0.5, label='Wording', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.title('Histogram of Content and Wording')
plt.show()
```




```
# scatterplot for prompt id = 39c16e

plt.scatter(mergedDF[mergedDF['prompt_id'] == '39c16e']['content'], mergedDF[mergedDF['prompt_id'] == '39c16e']['wording'], la
plt.xlabel('Contents')
plt.ylabel('Wording')
plt.title('Scatterplot of Content vs Wording')
plt.legend()
plt.show()
```




```
# Find the range for prompt_id = 39c16e
content_range = mergedDF[mergedDF['prompt_id'] == '39c16e']['content'].max() - mergedDF[mergedDF['prompt_id'] == '39c16e']['cc

print("Range of 'Content':", content_range)

wording_range = mergedDF[mergedDF['prompt_id'] == '39c16e']['wording'].max() - mergedDF[mergedDF['prompt_id'] == '39c16e']['wc

print("Range of 'Wording':", wording_range)

# find the correlation of content and wording for prompt_id = 39c16e

correlation = mergedDF[mergedDF['prompt_id'] == '39c16e']['content'].corr(mergedDF[mergedDF['prompt_id'] == '39c16e']['wording
print(f'Correlation between content and wording - {correlation}')
```

```
Range of 'Content': 5.630185556899
Range of 'Wording': 6.15001165042611
Correlation between content and wording - 0.8129110061670941
```

Section 4: Words in Good and Bad Essays (Q5, 10 points)

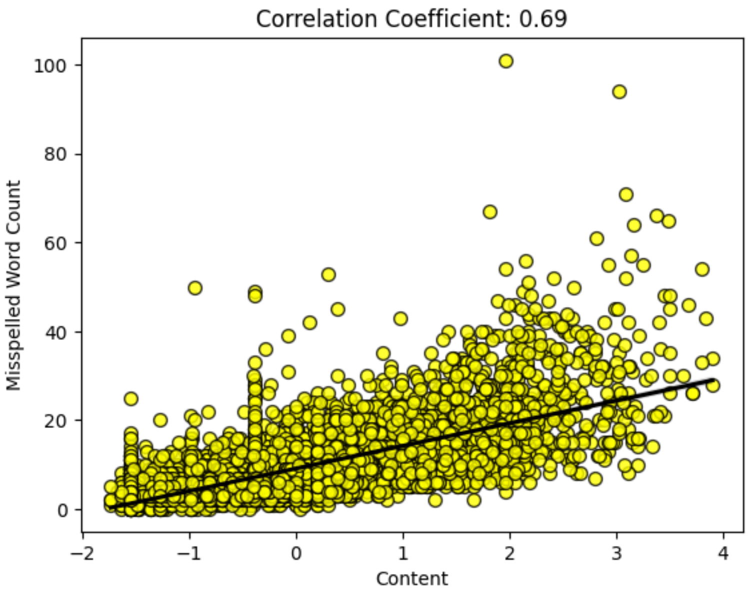
```
# Which words are over-represented in good essays (as per content and wording) while being under-represented in bad ones?
# Conversely, which words appear disproportionately in the bad essays?
# What is an appropriate statistic to use here? (10 points)

def label_essay(row):
    if row['content'] > mergedDF['content'].median() and row['wording'] > mergedDF['wording'].median():
```

```
        return "good"
    else:
        return "bad"

df['label'] = mergedDF.apply(label_essay, axis=1)

# df.head()

good_essays = mergedDF[df['label'] == 'good']
bad_essays = mergedDF[df['label'] == 'bad']

good_word_counts = Counter(" ".join(good_essays['text']).lower().split())
bad_word_counts = Counter(" ".join(bad_essays['text']).lower().split())

# print(good_word_counts)
# print(bad_word_counts)

overrepresented_in_good = [word for word, count in good_word_counts.items() if count > bad_word_counts.get(word, 0)]
overrepresented_in_good = [word for word in overrepresented_in_good if word.lower() not in nltk.corpus.stopwords.words('englis

overrepresented_in_bad = [word for word, count in bad_word_counts.items() if count > good_word_counts.get(word, 0)]
overrepresented_in_bad = [word for word in overrepresented_in_bad if word.lower() not in nltk.corpus.stopwords.words('english'

# print("Words overrepresented in 'good' essays:")
print(overrepresented_in_good)

# print("\nWords overrepresented in 'bad' essays:")
print(overrepresented_in_bad)

#  This shows that spelling errors are more in bad_essays. We can consider that as part of feature set to improve performace c
```

```
['third', 'wave', 'experimentto', 'see', 'people', 'new', 'one', 'leader', 'government.', 'gained', 'popularity', 'wantec
['particip', 'thos', 'contro.', 'thrid', 'sumbol', 'strenght', 'movement.thrid', 'take', 'mr', 'proved', 'croud', 'decit
```

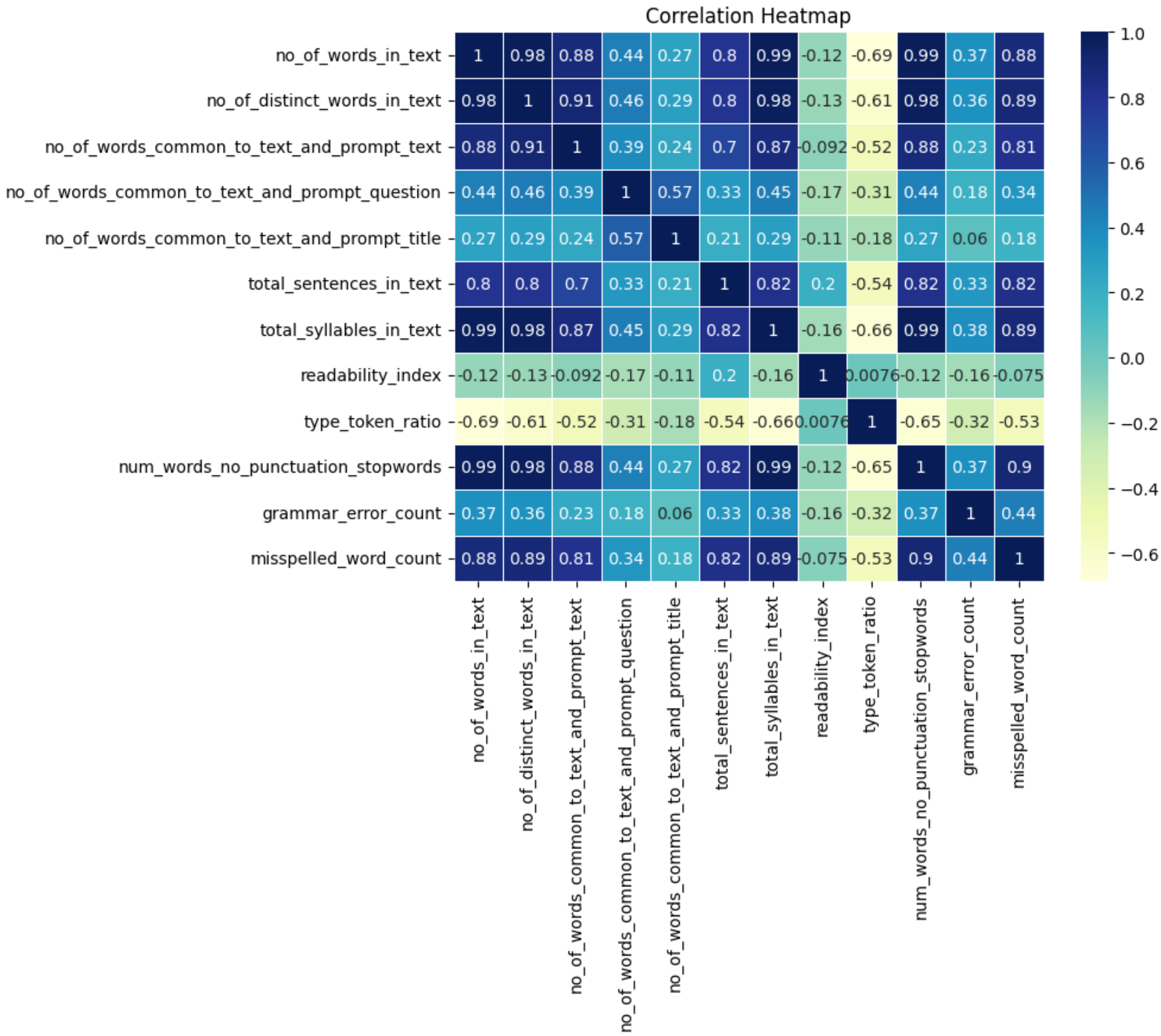
Section 5: Three Interesting Plots (Q6, 15 points)

```
# Create three plots of your own using the dataset that you think reveal something very interesting.
# Explain what it is, and anything else you learned from your exploration. (15 points)

# correlation coefficient of content and misspelled_word_count
correlation_coefficient = mergedDF['content'].corr(df['misspelled_word_count'])

# Create a scatter plot with a regression line
sns.regplot(x = mergedDF['content'], y = df['misspelled_word_count'], data=df, scatter_kws={"s": 50, "edgecolor":'black'}, col
plt.title(f'Correlation Coefficient: {correlation_coefficient:.2f}')
plt.xlabel('Content')
plt.ylabel('Misspelled Word Count')
plt.show()

# the grapth shows that content and missplled_word_count are strongly correlated.
```

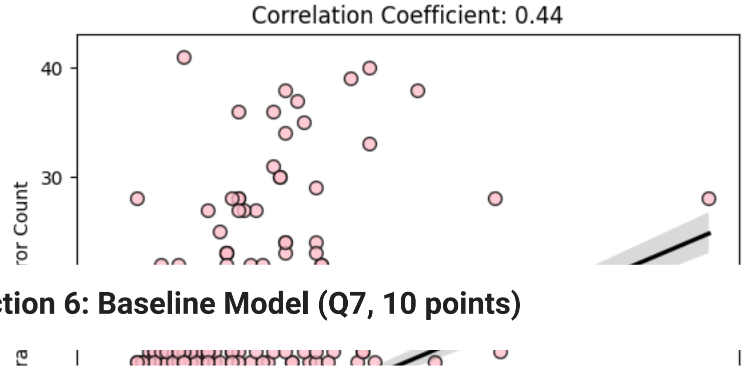



```
corr = df[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','no_of_words_comm
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','grammar_error_count','misspelled_word_count']].corr()
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='YlGnBu', linewidths=0.5)
plt.title("Correlation Heatmap")

plt.show()

# graph shows the correlation coefficient between all columns in the dataframe, this will help us design our model.
```




```
# correlation coefficient of content and misspelled_word_count
correlation_coefficient = df['misspelled_word_count'].corr(df['grammar_error_count'])

# Create a scatter plot with a regression line
sns.regplot(x = df['misspelled_word_count'], y = df['grammar_error_count'], data=df, scatter_kws={"s": 50, "edgecolor":'black'
plt.title(f'Correlation Coefficient: {correlation_coefficient:.2f}')
plt.xlabel('Misspelled Word Count')
plt.ylabel('Grammar Error Count')
plt.show()

# The graph shows that as missplled_word_count increases, the students are more likely to make spelling mistakes too.
```



Section 6: Baseline Model (Q7, 10 points)

```
# Now build a baseline model for this task. We will call this Model 0.
# You will train linear regression models for both content and wording on 80% of the training data and test it on the remainin
# Use only the original five features described above. Report the mean squared error of each model. What do you make of the er

X = df[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','no_of_words_common_
y = mergedDF['content']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 0.33457265041187195
```

```
X = df[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','no_of_words_common_
y = mergedDF['wording']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Error rate is relatively more because we haven't added highly corrleated feautures and we haven't done any preprocessing.
```

```
Mean Squared Error: 0.500696389487351
```

Section 7: Feature Cleaning and Additional Models (Q8 & Q9, 20 points total)

```
# The basic features as defined above are not really suited for the task.
# Features can be preprocessed (or cleaned) to improve them before feeding into the model (e.g. normalize them, do a special t
# This can significantly improve the performance of your model. Do preprocessing for all the features (the original five plus
# Explain what you did. (10 points)

# fill missing values
df['no_of_words_in_text'].fillna(df['no_of_words_in_text'].mean(), inplace=True)
df['no_of_distinct_words_in_text'].fillna(df['no_of_distinct_words_in_text'].mean(), inplace=True)
df['no_of_words_common_to_text_and_prompt_text'].fillna(df['no_of_words_common_to_text_and_prompt_text'].mean(), inplace=True)
df['no_of_words_common_to_text_and_prompt_question'].fillna(df['no_of_words_common_to_text_and_prompt_question'].mean(), inpla
df['no_of_words_common_to_text_and_prompt_title'].fillna(df['no_of_words_common_to_text_and_prompt_title'].mean(), inplace=Tru
df['total_sentences_in_text'].fillna(df['total_sentences_in_text'].mean(), inplace=True)
df['total_syllables_in_text'].fillna(df['total_syllables_in_text'].mean(), inplace=True)
df['readability_index'].fillna(df['readability_index'].mean(), inplace=True)
df['type_token_ratio'].fillna(df['type_token_ratio'].mean(), inplace=True)
df['num_words_no_punctuation_stopwords'].fillna(df['num_words_no_punctuation_stopwords'].mean(), inplace=True)
df['grammar_error_count'].fillna(df['grammar_error_count'].mean(), inplace=True)
df['misspelled_word_count'].fillna(df['misspelled_word_count'].mean(), inplace=True)


# Removing outliers

# Calculate Q1 and Q3
Q1 = mergedDF['content'].quantile(0.25)
Q3 = mergedDF['content'].quantile(0.75)
```

```
# Calculate the IQR
IQR = Q3 - Q1

# Define the upper and lower bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
filtered_df = mergedDF[(mergedDF['content'] >= lower_bound) & (mergedDF['content'] <= upper_bound)]

Q1 = mergedDF['wording'].quantile(0.25)
Q3 = mergedDF['wording'].quantile(0.75)

IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
filtered_df = mergedDF[(mergedDF['wording'] >= lower_bound) & (mergedDF['wording'] <= upper_bound)]
filtered_df_features = df[df['student_id'].isin(filtered_df['student_id'])]

# I have removed outliers, filled missing values with mean and normalized the data.


# For each of the two tasks (content and wording) create two models:
# Model 1 should use the cleaned features and linear regression for training. You can do some (potentially non-linear) scaling
# Model 2 should use the cleaned features and an algorithm other than logistic regression (e.g. Random Forest, Nearest Neighbo
# [Note: scikit-learn is a user-friendly library which is used to perform data loading, pre-processing, transformations, algor
# Compare their performance and explain your reasoning for the differences in their performances. (10 points)

# Model 1 for Content

X = filtered_df_features[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','n
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','grammar_error_count','misspelled_word_count']]
y = filtered_df['content']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

scaler_x = MinMaxScaler()
X_train_normalized = scaler_x.fit_transform(X_train)
X_test_normalized = scaler_x.transform(X_test)

scaler_y = MinMaxScaler()
y_train_normalized = scaler_y.fit_transform(y_train.values.reshape(-1, 1))

model = LinearRegression()
model.fit(X_train_normalized, y_train_normalized)

y_pred_normalized = model.predict(X_test_normalized)
y_pred = scaler_y.inverse_transform(y_pred_normalized)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error for Content: {mse}')

# Model 1 for Wording

X = filtered_df_features[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','n
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','misspelled_word_count','grammar_error_count']]
y = filtered_df['wording']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

scaler_x = MinMaxScaler()
X_train_normalized = scaler_x.fit_transform(X_train)
X_test_normalized = scaler_x.transform(X_test)

scaler_y = MinMaxScaler()
y_train_normalized = scaler_y.fit_transform(y_train.values.reshape(-1, 1))

model = LinearRegression()
model.fit(X_train_normalized, y_train_normalized)

y_pred_normalized = model.predict(X_test_normalized)
y_pred = scaler_y.inverse_transform(y_pred_normalized)

mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error for Wording: {mse}')
```

```
Mean Squared Error for Content: 0.2531063672413399
Mean Squared Error for Wording: 0.3815034979031959
```

```
# Random Forest
# Model 2 for Content

X = filtered_df_features[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','n
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','grammar_error_count','misspelled_word_count']]
y = filtered_df['content']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

rf_regressor = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=40)
rf_regressor.fit(X_train, y_train)

y_pred = rf_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error for Wording using Random Forest: {mse}")

# Model 2 for Wording

X = filtered_df_features[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','n
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','grammar_error_count','misspelled_word_count']]
y = filtered_df['wording']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

rf_regressor = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=40)
rf_regressor.fit(X_train, y_train)

y_pred = rf_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error for Wording using Random Forest: {mse}")

# Linear Regression is assuming a linear relationship between features and content/wording.
# Since the features don't seem to have linear relationship, non-linear model is performing better.
```

```
Mean Squared Error for Wording using Random Forest: 0.19270127166259574
Mean Squared Error for Wording using Random Forest: 0.3261538786071566
```

```
# K-Nearest Neighbours

# Model 3 for Content
X = filtered_df_features[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','n
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','grammar_error_count','misspelled_word_count']]
y = filtered_df['content']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

k = 24
regressor = KNeighborsRegressor(n_neighbors=k)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Model 3 for Wording
X = filtered_df_features[['no_of_words_in_text','no_of_distinct_words_in_text','no_of_words_common_to_text_and_prompt_text','n
        'no_of_words_common_to_text_and_prompt_title','total_sentences_in_text','total_syllables_in_text','readability_index',
        'num_words_no_punctuation_stopwords','grammar_error_count','misspelled_word_count']]
y = filtered_df['wording']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

k = 24
regressor = KNeighborsRegressor(n_neighbors=k)

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.2076733970484275
Mean Squared Error: 0.3633387184865767
```

Section 8: Kaggle Submission Screenshots (Q10, 5 points)

Public Score:

Private Score:

Kaggle profile link:

Screenshot(s):