Text Extraction:

Imports

In [1]:

```
from zipfile import ZipFile
from bs4 import BeautifulSoup

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import nltk

nltk.download('words')
nltk.download("cmudict")

from nltk import word_tokenize
from nltk import sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk import pos_tag
from nltk.corpus import cmudict

from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer, CountVecto

import string
```

```
[nltk_data] Downloading package words to
[nltk_data]     /Users/varadrajrameshpoojary/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package cmudict to
[nltk_data]     /Users/varadrajrameshpoojary/nltk_data...
[nltk_data]   Package cmudict is already up-to-date!
```

In [2]:

```
def read_file_from_zip(path="data/lang-8.zip"):
    """
    A generator function which reads html documents
    as raw text from the zip file

    Parameters
    ----------
    path : string
        path to the zip file

    Returns
    -------
    A dictionary of filename and raw text extracted
    from the file
    """
    archive = ZipFile(path, "r")

    for file in archive.namelist()[1:]:
        yield {
            "filename": file,#.removeprefix("lang-8/"),
            "data": archive.read(file)
        }
```

In [ ]:

In [3]:

```
def extract_data_from_file(path="data/lang-8.zip"):
    """
    A generator function which reads html files from zip
    and extracts text and native language from the raw
    text

    Parameters
    ----------
    path : string
        path to the zip file

    Returns
    -------
    A dictionary of extracted content and native language
    of the author
    """
    for data_dict in read_file_from_zip(path):
        soup = BeautifulSoup(data_dict["data"])

#         author = soup.find_all("p", attrs={"class": "spaced"})[1].get_text().strip()
        native_lang = soup.find("li", attrs={"data-title": "Native language"}).get_text().
        filename = data_dict["filename"]
        text = soup.find("div", attrs={"id": "body_show_ori"}).get_text().strip()

        preprocessed_data = {
            "text": text,
#             "author": author,
            "native_lang": native_lang,
            "filename": filename
        }

        yield preprocessed_data
```

In [4]:

```
# for i in extract_data_from_file():
#     print(i)
#     print()
```

Feature Extraction

Text Length

In [5]:

```
def get_text_length(text):
    """
    Returns the number of words in a text without punctuations.
    Counts clitics as separate words.

    Parameters
    ----------
    text : str
        A text from which we find the number of words

    Returns
    -------
    An int which represents the number of words in the text
    """
    non_punc = []
    for word in word_tokenize(text):
        if word not in string.punctuation:
```

```
            non_punc.append(word)
    return len(non_punc)
```

Lexical Density

In [6]:

```
def get_lexical_density(text):
    """
    Returns the lexical density of a text. That is the ratio of open class words.
    in the text

    Parameters
    ----------
    text : str
        A text from which we find the lexical density

    Returns
    -------
    A float which represents the lexical density
    """
    open_class_prefix = {"N", "V", "J", "R"}
    open_class_total = 0
    word_count = 0
    if len(text) == 0:
        return float(0)
    for word, pos in pos_tag(word_tokenize(text)):
        if word not in string.punctuation:
            word_count += 1
            if pos[0] in open_class_prefix:
                open_class_total += 1
    return open_class_total/word_count
```

Avg Sentence Length

In [7]:

```
def get_average_sentence_length(text):
    """
    Returns the average sentence length of a text. Does not count punctuations and counts

    Parameters
    ----------
    text : str
        A text from which we find the average sentence length

    Returns
    -------
    A float which represents the average sentence length
    """
    if len(text) == 0:
        return float(0)
    sent_lengths = 0
    for sentence in sent_tokenize(text):
        word_count = 0
        for word in word_tokenize(sentence):
            if word not in string.punctuation:
                word_count += 1
        sent_lengths += word_count
    return sent_lengths/len(sent_tokenize(text))
```

Avg Word Length

In [8]:

```
def get_average_word_length(text):
```

```
    """
    Returns the average sentence length of a text. Does not count punctuations
    and counts clitics.

    Parameters
    ----------
    text : str
        A text from which we find the average sentence length

    Returns
    -------
    A float which represents the average sentence length
    """
    if len(text) == 0:
        return float(0)
    word_count = 0
    lengths_sum = 0
    for word in word_tokenize(text):
        if word not in string.punctuation:
            lengths_sum += len(word)
            word_count += 1
    return lengths_sum/word_count
```

In [9]:

```
s0 = ""
s1 = """I went to the park today.
I love going there because I always have so much fun.
I invited some friends but they didn't come.
That's fine because I met a new person there.
He had a dog.
""" #40,
s2 = "I have so much work to do today. I am stressed" #11

# get_text_length
assert type(get_text_length(s0)) == int, "Must be an interger"
assert get_text_length(s0) == 0, "Empty string must return 0"
assert get_text_length(s1) == 40, "s1 has 40 words"
assert get_text_length(s2) == 11, "s2 has 11 words"
print("get_text_length tests pass")

assert type(get_lexical_density(s0))== float, "Must be a float"
assert get_lexical_density(s0) == 0, "Empty string must return 0"
assert get_lexical_density(s1) == 24/40, "24 open class words out of 40"
assert get_lexical_density(s2) == 8/11, "8 open class words out of 40"
print("get_lexical_density tests pass")

assert type(get_average_sentence_length(s0)) == float, "Must be a float"
assert get_average_sentence_length(s0) == 0, "Empty string must return 0"
assert get_average_sentence_length(s1) == 40/5, "40 words over the span of 5 sentences"
assert get_average_sentence_length(s2) == 11/2, "11 words over the span 2 sentences"
print("get_average_sentence_length tests pass")

assert type(get_average_word_length(s0)) == float, "Must be a float"
assert get_average_word_length(s0) == 0, "Empty string must return 0"
assert get_average_word_length(s1) == 142/40, "142 total characters spread across 40 words
assert get_average_word_length(s2) == 35/11, "35 character spread across 11 words"
print("get_average_word_length tests pass")
```

```
get_text_length tests pass
get_lexical_density tests pass
get_average_sentence_length tests pass
get_average_word_length tests pass
```

POS Count

In [10]:

```
def get_pos_count(text):
    """
    Counts the number of nouns, verbs and adjectives in a text.

    Parameters
    ----------
    text : str
        A text for which we find the number of nouns, verbs
        and adjectives

    Returns
    -------
    A tuple of (noun_count: int, verb_count: int, adj_count: int)
    which represents the number of nouns, verbs adjectives in the text
    respectively
    """
    noun_count = 0
    verb_count = 0
    adj_count = 0

    if len(text) == 0:
        return 0, 0, 0

    for word, pos in pos_tag(word_tokenize(text)):
        if(pos[0] == 'N'):
            noun_count += 1
        if(pos[0] == 'V'):
            verb_count += 1
        if(pos == 'JJ'):
            adj_count += 1
    return noun_count, verb_count, adj_count
```

In [11]:

```
s1 = """I went to the park today.
I love going there because I always have so much fun.
I invited some friends but they didn't come.
That's fine because I met a new person there.
He had a dog."""

s2 = """Chelsea English School is offering a Summer School Program in Iwaki, Fukushima, a
We will be hosted by "Namakiba" farm, an agricultural concern run by an Iwaki City coopera

assert get_pos_count(s1) == (6, 10, 3)
assert get_pos_count(s2) == (47, 17, 16)

print("get_pos_count tests pass")
```

```
get_pos_count tests pass
```

OOV Words

In [12]:

```
def get_num_ovv_words(text):
    """
    Gets the number of out-of-vocabulary words in a text.

    Parameters
    ----------
    text : str
        A text for which we find the number of out-of-vocabulary
        words is to be found

    Returns
    -------
    The number of oov words in the text
```

```
    """
    text_vocab = set(w.lower() for w in text.split() if w.isalpha())
    english_vocab = set(w.lower() for w in nltk.corpus.words.words())
    ovv_words = text_vocab - english_vocab

    return len(ovv_words)
```

In [13]:

```
# get_num_ovv_words
s0 = ""
s1 = """ I haddd to leaasve earliae since yesterday was so tired.
And then I met you.
"""
s2 = "I have so much work to do today. I am stressseed"
assert type(get_num_ovv_words(s0)) == int, "Must be an interger"
assert get_num_ovv_words(s0) == 0, "Empty string must return 0"
assert get_num_ovv_words(s1) == 3, "s1 has 3 words out of vaocab"
assert get_num_ovv_words(s2) == 1, "s2 has 1 words out of vocab"
print("get_num_ovv_words tests pass")
```

```
get_num_ovv_words tests pass
```

Reading Ease

In [14]:

```
# Code adapted from lab

vowels = {"a","e","i","o","u","y"}
p_dict = cmudict.dict()

def get_reading_ease(text):
    """Returns the reading ease for a text.

    Parameters
    ----------
    text : str
       A text for which we find the reading ease.

    Returns
    -------
    reading_ease : float
        The reading ease for the text
    """
    syllable_count = 0
    word_count = 0

    for word in word_tokenize(text):
        if word not in string.punctuation:
            word_count += 1
            if word in p_dict:
                for pron in p_dict[word][0]:
                    if pron[-1] in ['0','1','2']:
                        syllable_count +=1
            else:
                for j in range(0,len(word)):
                    if word[j].lower() in vowels:
                         syllable_count= syllable_count+1

    reading_ease = (206.835 - (1.015*(word_count/len(sent_tokenize(text))))- (84.6*(syllak
    return reading_ease
```

In [15]:

```
assert 100 < get_reading_ease("I am done, man") < 140
assert -60 < get_reading_ease("Felicitations for achieving a thoroughly excellent resoluti
print("get_reading_ease tests pass")
```

get_reading_ease tests pass

Punctuation Counts

In [16]:

```
def get_punctuations_count(text):
    """
    Returns the number of punctuations in a text.

    Parameters
    ----------
    text : str
        A text for which we find the number of punctuations present

    Returns
    -------
    punct_count: int
                 An integer which represents the number of punctuations in the text
    """
    punct_count = 0
    if len(text) == 0:
        return 0
    for word in word_tokenize(text):
        if word in string.punctuation:
            punct_count += 1
    return punct_count
```

In [17]:

```
s1 = """I went to the park today.
I love going there because I always have so much fun.
I invited some friends but they didn't come.
That's fine because I met a new person there.
He had a dog."""

s2 = """Chelsea English School is offering a Summer School Program in Iwaki, Fukushima, a
We will be hosted by "Namakiba" farm, an agricultural concern run by an Iwaki City coopera


assert get_punctuations_count(s1) == 5
assert get_punctuations_count(s2) == 16

print("get_punctuations_count tests pass")
```

get_punctuations_count tests pass

Type-Token Ratio

In [18]:

```
def get_type_token_ratio(text):
    """
    Calculate type-token ratio from the text using the first
    num_words tokens

    Parameters
    ----------
    text : str
        A text for which we find the type-token ratio

    Returns
    -------
    type_token_ratio: int
                      An integer which represents the type token ratio for a given text
    """
    words = text.split()
```

```
    num_words = 100
    type_set = set(word.lower() for word in words[:num_words])
    return len(type_set) / num_words
```

In [19]:

```
s1 = """Chelsea English School is offering a Summer School Program in Iwaki, Fukushima, a
We will be hosted by "Namakiba" farm, an agricultural concern run by an Iwaki City coopera

s2 = """I'd like to acquire this skill, however it doesn't really fit into my schedule rig
assert get_type_token_ratio(s1) == 0.74
assert get_type_token_ratio(s2) == 0.48

print("get_type_token_ratio tests pass")
```

```
get_type_token_ratio tests pass
```

Asian Context

In [20]:

```
def get_asian_context_feature(text):
    """
    Return a binary value based on whether asian journal's context based words are present

    Parameters
    ----------
    text : str
        A text for which we find the presence of the words

    Returns
    -------
    value : boolean
            0 represents that the no word in the list is present in the text and 1 represe
    """

    lemmatizer = WordNetLemmatizer()
    with open("data/asian_words.txt", "r") as file:
        asian_journals_context_words = file.read().split("\n")

    for word in word_tokenize(text):
        if lemmatizer.lemmatize(word) in asian_journals_context_words:
            return 1
    return 0
```

In [21]:

```
asian_journal_test = """Frank moved to Guangzhou after long time consideration from anothe
I have not tried Indonesian food before but similar ones such as Singaporean and Malaysian
Ok, back to the Indonesian restaurant. It is a tradtional one decorating with local stuff,

asian_journal_test_2 = """Today I had TV conference with Malaysian in English.I know we Ja

european_journal_test = """I am missing the friends, and I will miss my life in US. It has

european_journal_test_2 = """A few days ago, I've discovered something pretty awesome: it
It could be decribed as the art of writing. After much practice, the results are really gr

assert get_asian_context_feature(asian_journal_test) == 1
assert get_asian_context_feature(european_journal_test) == 0
assert get_asian_context_feature(asian_journal_test_2) == 1
assert get_asian_context_feature(european_journal_test_2) == 0

print("get_asian_context_feature tests pass")
```

```
get_asian_context_feature tests pass
```

Word Importance

In [22]:

```
def get_word_importance(text):
    """This is a helper function to generate TF IDF scores for words present in the input
    count = CountVectorizer(stop_words='english', analyzer='word')
    word_count = count.fit_transform(text)

    tfidf_transformer = TfidfTransformer(smooth_idf=True, use_idf=True)
    tfidf_transformer.fit(word_count)
    tf_idf_vector = tfidf_transformer.transform(word_count)

    feature_names = count.get_feature_names_out()
    first_document_vector = tf_idf_vector[0]

    df_tfifd = pd.DataFrame(first_document_vector.T.todense(), index=feature_names, columr
    df_tfifd = df_tfifd.sort_values(by=["tfidf"], ascending=False)

    return df_tfifd
```

In [23]:

```
def get_top_important_words():
    """This is a helper function to fetch words with top TF IDF scores for both asian and

    no_of_top_words = 30

    asian_lang_text = []
    non_asian_lang_text = []

    lemmatizer = WordNetLemmatizer()
    asian_lang = ["Korean", "Japanese", "Mandarin Chinese"]

    for extracted_data in extract_data_from_file():
        if extracted_data['native_lang'] in asian_lang:
            asian_lang_text.append(extracted_data['text'] + ", ")
        else:
            non_asian_lang_text.append(extracted_data['text'] + ", ")

    asian_imp_words_df = get_word_importance(asian_lang_text)
    non_asian_imp_words_df = get_word_importance(non_asian_lang_text)

    asian_imp_words = asian_imp_words_df.index[:no_of_top_words]
    asian_imp_words_lem = [lemmatizer.lemmatize(x) for x in asian_imp_words]

    non_asian_imp_words = non_asian_imp_words_df.index[:no_of_top_words]
    non_asian_imp_words_lem = [lemmatizer.lemmatize(x) for x in non_asian_imp_words]

    return asian_imp_words_lem, non_asian_imp_words_lem
```

In [24]:

```
asian_imp_words_index, non_asian_imp_words_index = get_top_important_words()
```

Asian Important Words

In [25]:

```
def get_imp_words_asian_feature(text):
    """
    Return a binary value based on whether asian journal's important words (based on TF-II

    Parameters
    ----------
    text : str
        A text for which we find the presence of the words
```

```
    Returns
    -------
    value : boolean
            0 represents that the no word in the list is present in the text and 1 represe
    """
    lemmatizer = WordNetLemmatizer()

    for word in word_tokenize(text):
        if lemmatizer.lemmatize(word) in asian_imp_words_index:
            return 1

    return 0
```

Non-Asian Important Words

In [26]:

```
def get_imp_words_non_asian_feature(text):
    """
    Return a binary value based on whether european journal's important words (based on TF

    Parameters
    ----------
    text : str
        A text for which we find the presence of the words

    Returns
    -------
    value : boolean
            0 represents that the no word in the list is present in the text and 1 represe
    """
    lemmatizer = WordNetLemmatizer()

    for word in word_tokenize(text):
        if lemmatizer.lemmatize(word) in non_asian_imp_words_index:
            return 1
    return 0
```

In [27]:

```
test_text_asian_feature = """Hi, my name is Sebastián. This is the first time I write here
I'm studying English and Japanese. I use the computer and internet a lot for studying, whi
As for English, its pronunciation is hard, but reading isn't that much hard. Besides, on i
On the other side, reading Japanese is something that takes lots of time and effort. Even
As I said above, the use of the computer and internet has helped me a lot. I think this is
What do you think?"""

test_text_non_asian_feature = """A few days ago, I've discovered something pretty awesome:
It could be decribed as the art of writing. After much practice, the results are really gr


assert get_imp_words_asian_feature(test_text_asian_feature) == 1
assert get_imp_words_non_asian_feature(test_text_asian_feature) == 0
assert get_imp_words_non_asian_feature(test_text_non_asian_feature) == 1
assert get_imp_words_asian_feature(test_text_non_asian_feature) == 0

print("get_imp_words_asian_feature tests pass")
print("get_imp_words_non_asian_feature tests pass")
```

```
get_imp_words_asian_feature tests pass
get_imp_words_non_asian_feature tests pass
```

In [ ]:

In [28]:

```
def get_document_list(txt_path):
```

```python
    """
    Extracts the list of documents stores in a text file

    Parameters
    ----------
    text_path : str
        The string defining path of the text document

    Returns
    -------
    A list of filenames extracted from the file
    """
    doc_list = []

    with open(txt_path, "r") as f:
        for filename in f.readlines():
            doc_list.append(filename.strip())

    return doc_list
```

In [29]:

```python
def extract_all_features(
    txt_path, csv_path, zip_path="data/lang-8.zip", verbose=False
):
    """
    Reads the zip file from path, extracts features from
    preprocessed text and combines them together to save
    them to a csv file

    Parameters
    ----------
    csv_path : str
            path at which the generated csv file is to be saved
    zip_path : str
            path to the zip file
    verbose : boolean
            specify whether to print the processed filename or not
    """

    # Return if the file already exists
    if os.path.isfile(csv_path):
        return

    # Lists of relevant features
    asian_lang = ["Korean", "Japanese", "Mandarin Chinese"]
    names = []
    text_lens = []
    lexical_densities = []
    avg_sent_lens = []
    avg_word_lens = []
    oov_word_counts = []

    reading_eases = []
    punctuations_counts = []
    type_token_ratios = []
    asian_context_features = []
    imp_words_asian_features = []
    imp_words_non_asian_features = []

    noun_counts, verb_counts, adj_counts = [], [], []
    targets = []

    # Lists of training, validation and test files
    doc_list = get_document_list(txt_path)
```

```
    for extracted_data in extract_data_from_file(zip_path):

        if extracted_data["filename"].removeprefix("lang-8/") not in doc_list:
            continue

        if extracted_data["native_lang"] == "Russian":
            continue

        if extracted_data['native_lang'] in asian_lang:
            target = 1
        else:
            target = 0

        targets.append(target)
        names.append(extracted_data['filename'][7:-5])
        text_lens.append(get_text_length(extracted_data['text']))
        lexical_densities.append(get_lexical_density(extracted_data['text']))
        avg_sent_lens.append(get_average_sentence_length(extracted_data['text']))
        avg_word_lens.append(get_average_word_length(extracted_data['text']))
        oov_word_counts.append(get_num_ovv_words(extracted_data['text']))

        reading_eases.append(get_reading_ease(extracted_data['text']))
        punctuations_counts.append(get_punctuations_count(extracted_data['text']))
        type_token_ratios.append(get_type_token_ratio(extracted_data['text']))
        asian_context_features.append(get_asian_context_feature(extracted_data['text']))
        imp_words_asian_features.append(get_imp_words_asian_feature(extracted_data['text']
        imp_words_non_asian_features.append(get_imp_words_non_asian_feature(extracted_data

        noun_count, verb_count, adj_count = get_pos_count(extracted_data['text'])
        noun_counts.append(noun_count)
        verb_counts.append(verb_count)
        adj_counts.append(adj_count)

        if verbose:
            print(len(targets), extracted_data["filename"])

    feature_df = pd.DataFrame(
        np.array([
            names,
            text_lens,
            lexical_densities,
            avg_sent_lens,
            avg_word_lens,
            oov_word_counts,

            reading_eases,
            punctuations_counts,
            type_token_ratios,
            asian_context_features,
            imp_words_asian_features,
            imp_words_non_asian_features,

            noun_counts,
            verb_counts,
            adj_counts,
            targets
        ]).T,
        columns=[
            "filename",
            "text_length",
            "lexical_density",
            "average_sentence_length",
            "average_word_length",
            "oov_word_counts",

            "reading_ease",
```

```
            "punctuation_count",
            "type_token_ratio",
            "asian_context_feature",
            "asian_imp_word",
            "non_asian_imp_word",

            "noun_counts",
            "verb_counts",
            "adjective_counts",
            "target_region"
        ])

    feature_df.to_csv(csv_path)
```

In [30]:

```
def create_train_dev_test_csvs(paths={
                                    "data/train.txt": "data/train.csv",
                                    "data/dev.txt": "data/dev.csv",
                                    "data/test.txt": "data/test.csv"
                                },
                                zip_path="data/lang-8.zip"):
    """
    Takes in paths of text documents containing filenames from which
    information is to be extracted, extracts informtion from them and
    store them as csvs for train, validation and test

    Parameters
    ----------
    paths : dict
        a dictionary with keys as paths for text documents to read filenames
        and values as paths for the csv documents to save extracted features
    zip_path : str
            path to the zip file
    """

    for txt_path, csv_path in paths.items():
        extract_all_features(txt_path, csv_path, zip_path)
```

In [31]:

```
def read_csvs(train, val, test):
    """
    Reads train, validation and test sets from disk

    Parameters
    ----------
    train : str
        The path of the training csv file
    train : str
        The path of the training csv file
    train : str
        The path of the training csv file

    Returns
    -------
    A tuple of train, validation and test dataframes
    """
    train_csv = None
    val_csv = None
    test_csv = None

    try:
        train_csv = pd.read_csv(train)
    except:
        pass
    try:
```

```
        val_csv = pd.read_csv(val)
    except:
        pass
    try:
        test_csv = pd.read_csv(test)
    except:
        pass

    return train_csv, val_csv, test_csv
```

In [32]:

```
# Reading the data

train_csv_path = r"data/train.csv"
val_csv_path = r"data/dev.csv"
test_csv_path = r"data/test.csv"

train_txt_path = r"data/train.txt"
val_txt_path = r"data/dev.txt"
test_txt_path = r"data/test.txt"

paths = {
    train_txt_path: train_csv_path,
    val_txt_path: val_csv_path,
    test_txt_path: test_csv_path
}

zip_path = r"data/lang-8.zip"

if not (os.path.isfile(
    train_csv_path
) and os.path.isfile(
    val_csv_path
) and os.path.isfile(
    test_csv_path
)):
    create_train_dev_test_csvs(paths)

train_data, val_data, test_data = read_csvs(train_csv_path, val_csv_path, test_csv_path)
```

Exploratory Data Analysis

In [33]:

```
train_data = train_data.drop(columns=["Unnamed: 0", "filename"])
train_data
```

Out[33]:

	text_length	lexical_density	average_sentence_length	average_word_length	oov_word_counts	reading_
0	281	0.480427	70.250000	4.259786	21	18.71
1	29	0.655172	29.000000	4.448276	1	46.12
2	307	0.592834	38.375000	4.566775	17	37.81
3	67	0.611940	22.333333	4.104478	5	57.89
4	25	0.520000	25.000000	3.520000	0	79.94
...	...	...	...	...	...	
738	107	0.532710	53.500000	4.130841	6	29.98
739	64	0.671875	32.000000	4.296875	0	48.77
740	63	0.523810	21.000000	3.952381	3	70.03

	text_length	lexical_density	average_sentence_length	average_word_length	oov_word_counts	reading_
741	37	0.702703	37.000000	4.567568	2	29.80
742	146	0.547945	7.300000	4.116438	13	69.62

743 rows × 15 columns

In [34]:

```
val_data = val_data.drop(columns=["Unnamed: 0", "filename"])
val_data
```

Out[34]:

	text_length	lexical_density	average_sentence_length	average_word_length	oov_word_counts	reading_
0	699	0.595136	14.265306	4.214592	40	69.14
1	618	0.600324	22.888889	4.872168	34	44.10
2	83	0.590361	20.750000	4.421687	5	53.26
3	32	0.625000	32.000000	4.406250	1	50.09
4	70	0.585714	23.333333	4.828571	4	50.20
...	...	...	...	...	...	
241	73	0.602740	12.166667	3.849315	1	80.91
242	88	0.579545	12.571429	3.829545	2	82.55
243	37	0.648649	18.500000	4.810811	0	41.72
244	322	0.562112	21.466667	4.760870	14	57.09
245	198	0.555556	33.000000	4.287879	6	44.30

246 rows × 15 columns

In [35]:

```
test_data = test_data.drop(columns=["Unnamed: 0", "filename"])
test_data
```

Out[35]:

	text_length	lexical_density	average_sentence_length	average_word_length	oov_word_counts	reading_
0	47	0.638298	11.750000	3.957447	0	79.70
1	73	0.671233	73.000000	4.575342	4	-8.64
2	52	0.634615	17.333333	3.692308	1	67.22
3	80	0.650000	80.000000	4.337500	7	9.31
4	187	0.572193	31.166667	4.181818	7	62.09
...	...	...	...	...	...	
247	47	0.787234	47.000000	5.425532	2	0.73
248	39	0.615385	39.000000	4.282051	1	39.26
249	19	0.842105	19.000000	5.789474	2	4.99
250	110	0.600000	22.000000	4.709091	6	55.29
251	328	0.628049	65.600000	4.317073	28	19.79

252 rows × 15 columns

In [36]:

```python
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 743 entries, 0 to 742
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   text_length              743 non-null    int64  
 1   lexical_density          743 non-null    float64
 2   average_sentence_length  743 non-null    float64
 3   average_word_length      743 non-null    float64
 4   oov_word_counts          743 non-null    int64  
 5   reading_ease             743 non-null    float64
 6   punctuation_count        743 non-null    int64  
 7   type_token_ratio         743 non-null    float64
 8   asian_context_feature    743 non-null    int64  
 9   asian_imp_word           743 non-null    int64  
 10  non_asian_imp_word       743 non-null    int64  
 11  noun_counts              743 non-null    int64  
 12  verb_counts              743 non-null    int64  
 13  adjective_counts         743 non-null    int64  
 14  target_region            743 non-null    int64  
dtypes: float64(5), int64(10)
memory usage: 87.2 KB
```

In [37]:

```python
train_data.describe().T
```

Out[37]:

	count	mean	std	min	25%	50%	75%
text_length	743.0	108.492598	117.516712	1.000000	41.500000	74.000000	133.000000
lexical_density	743.0	0.614475	0.092623	0.333333	0.558528	0.600000	0.646110
average_sentence_length	743.0	30.431251	32.568711	1.000000	13.000000	21.000000	35.000000
average_word_length	743.0	4.433092	2.683678	2.428571	3.916667	4.191489	4.541325
oov_word_counts	743.0	6.328398	9.106281	0.000000	2.000000	4.000000	7.000000
reading_ease	743.0	53.392032	42.705474	-330.054665	40.071008	60.056250	74.395216
punctuation_count	743.0	12.671602	14.992625	0.000000	4.000000	8.000000	16.000000
type_token_ratio	743.0	0.499892	0.223763	0.010000	0.320000	0.530000	0.700000
asian_context_feature	743.0	0.169583	0.375519	0.000000	0.000000	0.000000	0.000000
asian_imp_word	743.0	0.492598	0.500282	0.000000	0.000000	0.000000	1.000000
non_asian_imp_word	743.0	0.344549	0.475541	0.000000	0.000000	0.000000	1.000000
noun_counts	743.0	26.371467	30.641556	0.000000	9.000000	17.000000	32.000000
verb_counts	743.0	21.301480	22.888175	0.000000	9.000000	15.000000	26.000000
adjective_counts	743.0	9.053836	9.385991	0.000000	3.000000	6.000000	11.500000
target_region	743.0	0.335128	0.472353	0.000000	0.000000	0.000000	1.000000

Data Imbalance

In [38]:

```python
test_data["target_region"].value_counts(normalize=True)
```

Out[38]:

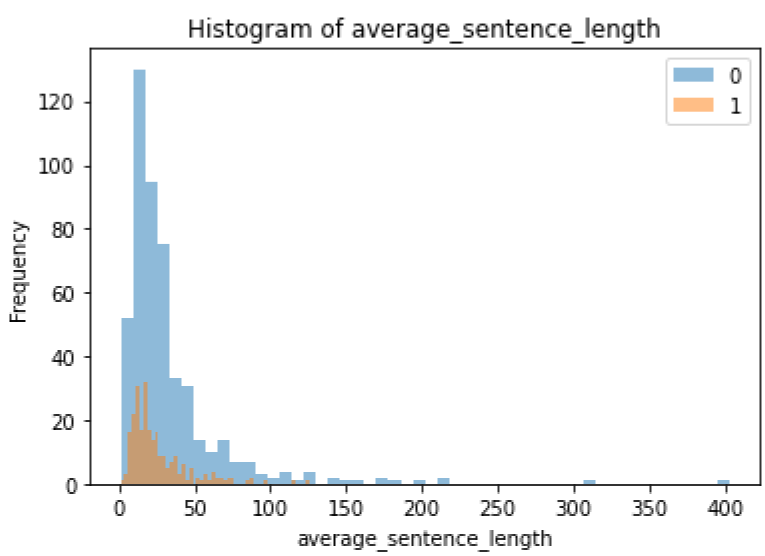
```
0    0.662698
1    0.337302
Name: target_region, dtype: float64
```

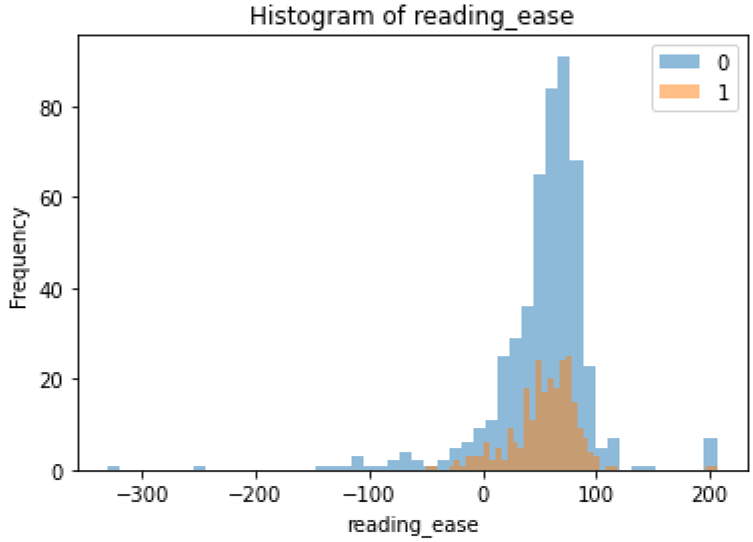
In [39]:
```
train_data.columns
```

Out[39]:
```
Index(['text_length', 'lexical_density', 'average_sentence_length',
       'average_word_length', 'oov_word_counts', 'reading_ease',
       'punctuation_count', 'type_token_ratio', 'asian_context_feature',
       'asian_imp_word', 'non_asian_imp_word', 'noun_counts', 'verb_counts',
       'adjective_counts', 'target_region'],
      dtype='object')
```

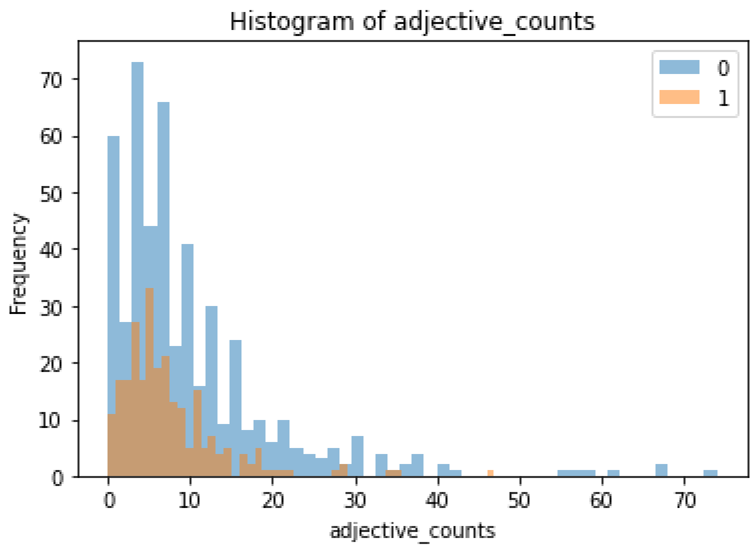
In [40]:
```
# Selecting feature types

binary_features_org = [
    "asian_context_feature",
    "asian_imp_word",
    "non_asian_imp_word"
]

target = "target_region"

numeric_features_org = list(
    set(train_data.columns)
    - set(binary_features_org)
    - set([target])
)

assert train_data.columns.shape[0] == len(
    binary_features_org
    + numeric_features_org
    + [target]
)
```

In [41]:
```
for i in numeric_features_org:
    feat = i
    ax = train_data.groupby("target_region")[feat].plot.hist(bins=50, alpha=0.5, legend=Tr
    plt.xlabel(feat)
    plt.title("Histogram of " + feat)
    plt.show()
```

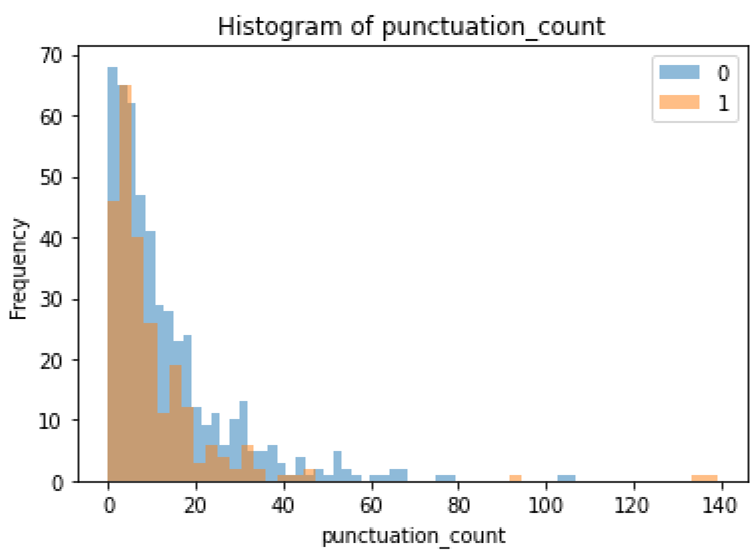




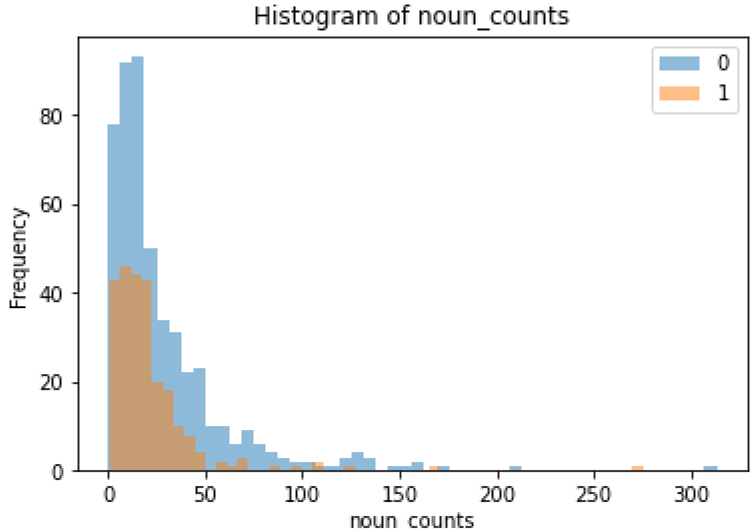
Histogram of reading_ease
Frequency
reading_ease
0
1
0
20
40
60
80
−300
−200
−100
0
100
200

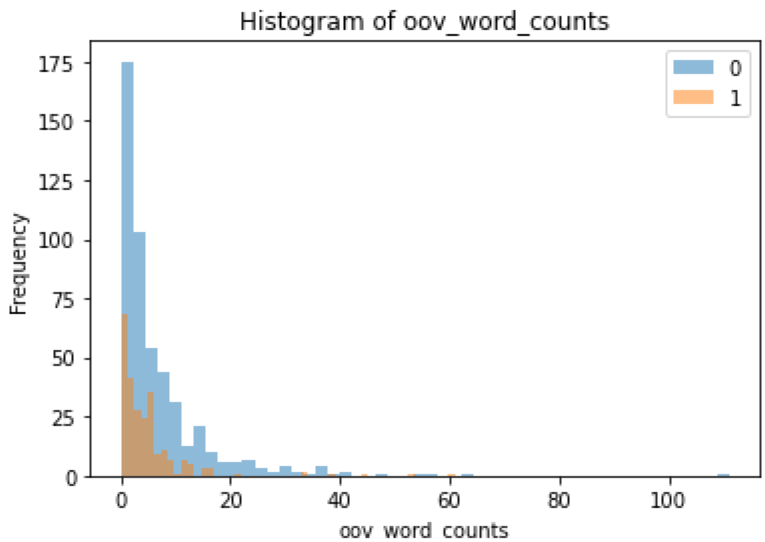


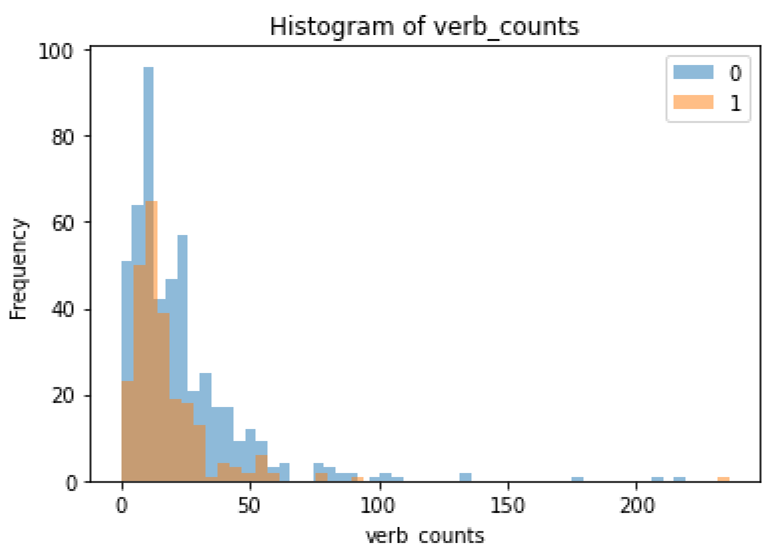
Histogram of adjective_counts
Frequency
adjective_counts
0
1
0
10
20
30
40
50
60
70

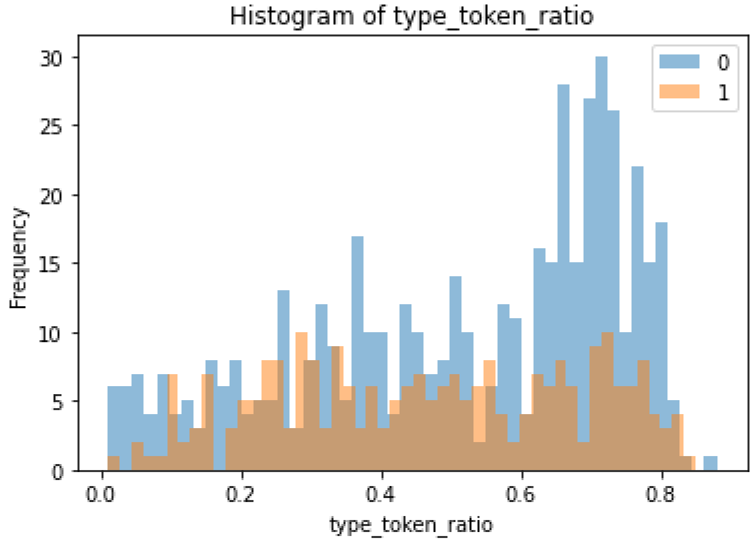


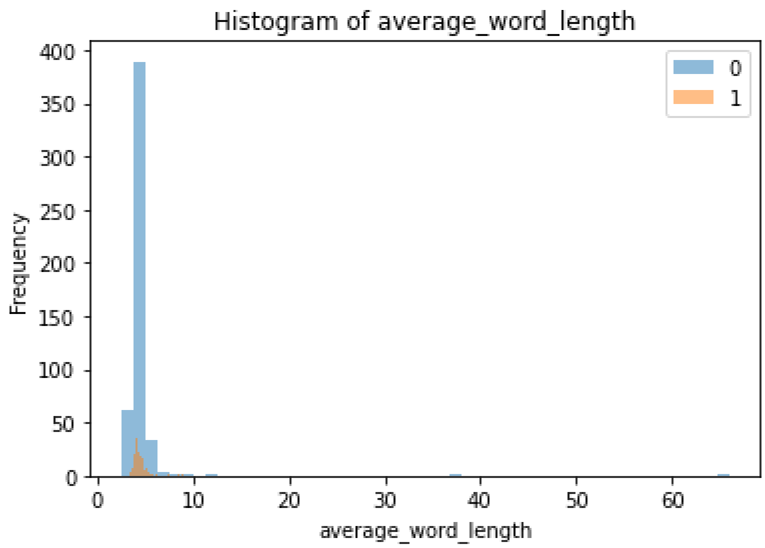
Histogram of punctuation_count
Frequency
punctuation_count
0
1
0
10
20
30
40
50
60
70
0
20
40
60
80
100
120
140

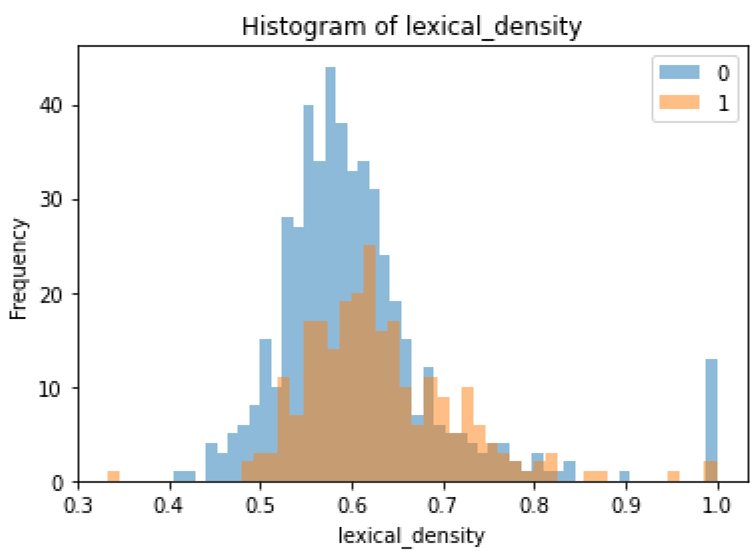

Histogram of noun_counts
Frequency
noun_counts
0
1
0
20
40
60
80
0
50
100
150
200
250
300

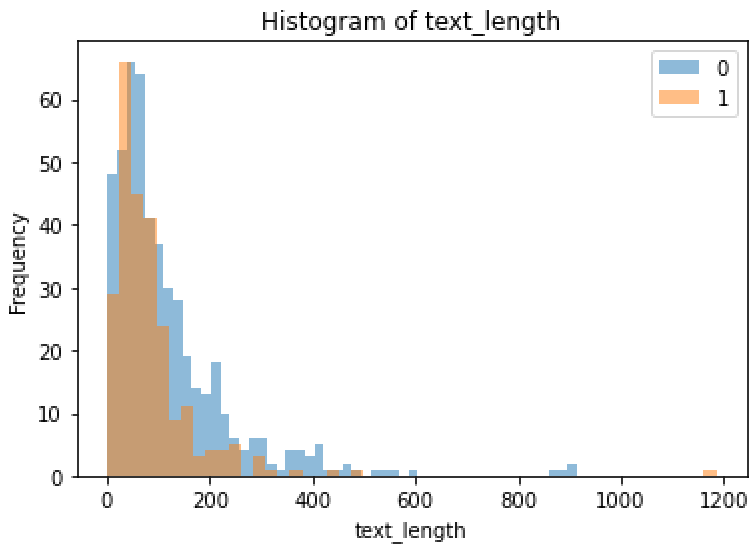

Histogram of oov_word_counts
Frequency
oov_word_counts
0
1
0
25
50
75
100
125
150
175
0
20
40
60
80
100


Histogram of verb_counts
Frequency
verb_counts
0
1
0
20
40
60
80
100
0
50
100
150
200


Histogram of type_token_ratio
Frequency
type_token_ratio
0
1


Histogram of average_word_length
Frequency
average_word_length
0
1


Histogram of lexical_density
Frequency
lexical_density
0
1



Interpretation

From the above histograms, we observe that several features have some differences between the two groups. Visually, we observe a difference in average sentence length where the European L1 speakers are writing longer sentences. We also observe that there is a greater variance for the lexical density between the 2 groups for example for lexical density between around 0.45 to 0.47 there is a clear distinction that the speaker is European.

In [42]:

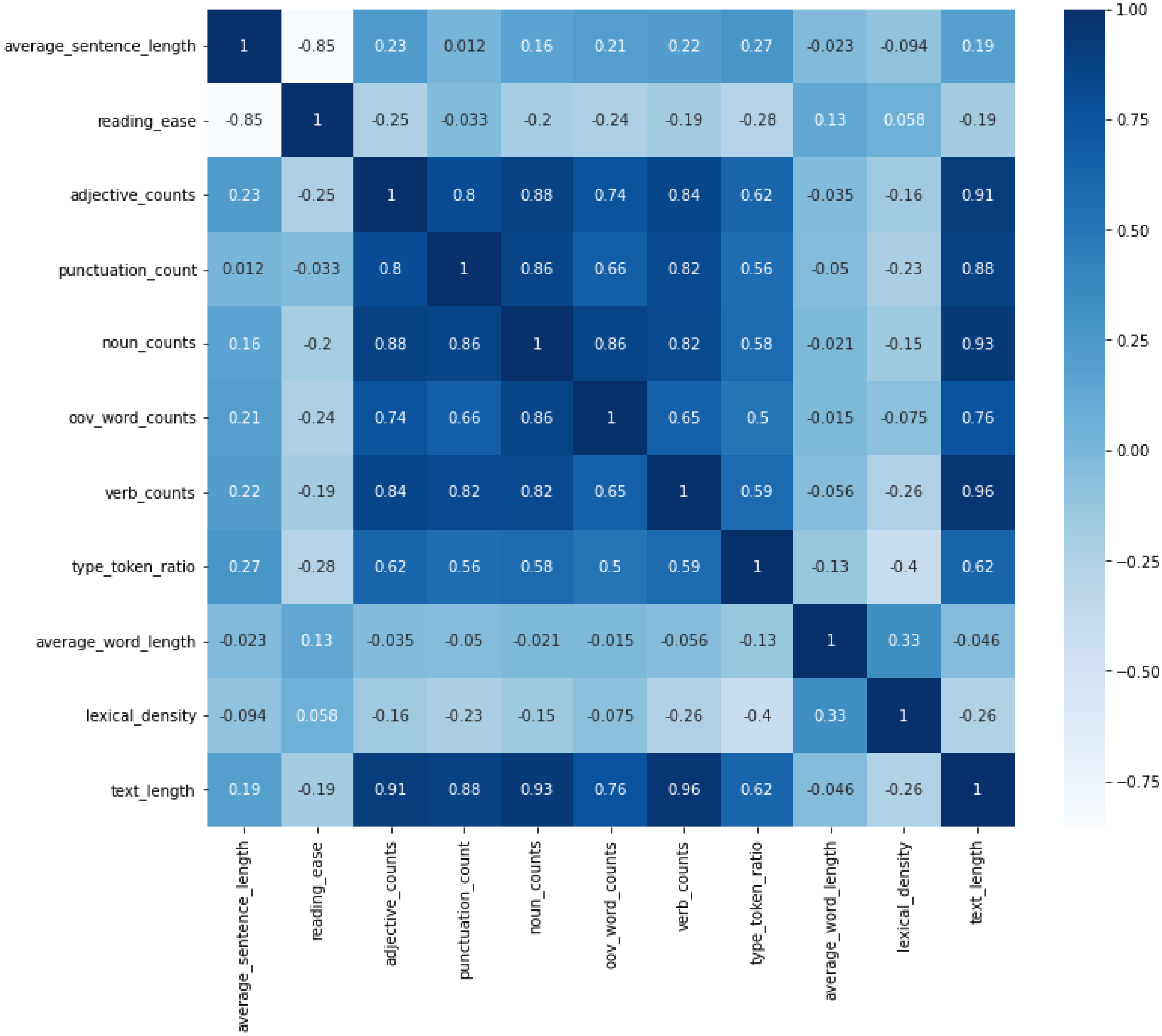
```
cor = train_data[numeric_features_org].corr()
cor
```

Out[42]:

	average_sentence_length	reading_ease	adjective_counts	punctuation_count	noun
average_sentence_length	1.000000	-0.854560	0.227307	0.011807	(
reading_ease	-0.854560	1.000000	-0.245517	-0.032827	-(
adjective_counts	0.227307	-0.245517	1.000000	0.800588	(
punctuation_count	0.011807	-0.032827	0.800588	1.000000	0
noun_counts	0.160695	-0.201604	0.876795	0.856934	1
oov_word_counts	0.208865	-0.244450	0.742415	0.664078	0
verb_counts	0.217404	-0.187747	0.841040	0.818669	(
type_token_ratio	0.268632	-0.284426	0.617158	0.563041	(
average_word_length	-0.023097	0.130178	-0.035431	-0.049673	-(
lexical_density	-0.094276	0.057953	-0.159322	-0.228157	-(
text_length	0.190506	-0.185584	0.909393	0.880017	(

In [43]:

```
plt.figure(figsize=(12, 10))
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues)
plt.show()
```



Interpretation

Because the text features are derived from a single text document, the majority of them are correlated to each other, meaning that removing one will affect the importance of the other features in a linear model. So, in order to complete the classification assignment, a viable choice is to employ a tree-based model.

Classification

Imports

In [44]:

```
from sklearn.dummy import DummyClassifier

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.preprocessing import (
    OneHotEncoder,
    PolynomialFeatures,
    StandardScaler,
)

from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import average_precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score

from sklearn.compose import make_column_transformer

from sklearn.pipeline import make_pipeline

from sklearn.feature_selection import RFE
from sklearn.preprocessing import PolynomialFeatures

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, Ridge
from catboost import CatBoostClassifier
from lightgbm.sklearn import LGBMClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier

from sklearn.metrics import precision_recall_curve
```

Splitting Data

In [45]:

```
# Splitting explanatory and target variables
target = "target_region"
X_train, y_train = train_data.drop(columns=[target]), train_data[target]
X_val, y_val = val_data.drop(columns=[target]), val_data[target]
X_test, y_test = test_data.drop(columns=[target]), test_data[target]
```

In [ ]:

In [46]:

```
results = {}
```

In [47]:

```
scoring_metrics = [
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
]
```

In [48]:

```
# Adapted from DSCI 573 Lecture Notes:

def cross_validate_model(model, X_train, y_train,
                         X_val, y_val, scoring, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data
    X_val : numpy array or pandas DataFrame
        X in the validation data
    y_val :
```

```python
        y in the validation data

    Returns
    ----------
        pandas Series with train and validation scores
    """

    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_val_pred = model.predict(X_val)

    scores = []

    for scoring_function in scoring:
        scores.append(scoring_function(y_train, y_train_pred))
        scores.append(scoring_function(y_val, y_val_pred))

    scores = pd.Series(
        data=scores,
        index=["train_accuracy",
               "test_accuracy",
               "train_precision",
               "test_precision",
               "train_recall",
               "test_recall",
               "train_f1",
               "test_f1"]
    )

    return scores
```

Eliminating Features

In [49]:

```python
def select_features(model,
                    X_train,
                    y_train,
                    X_val,
                    y_val,
                    numeric_features,
                    binary_features,
                    n_feats_to_drop=1,
                    verbose=False):
    """
    Eliminates least important features from the dataset

    Parameters
    ----------
    model : scikit-learn model
        ML model to decide the most important features
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data
    X_val : numpy array or pandas DataFrame
        X in the validation data
    y_val :
        y in the validation data
    numeric_features : list
        Original list of Numeric Features
    binary_features : list
        Original list of Binary Features
    n_cols_to_drop : int
        Number of least important columns to drop
```

```python
    verbose : boolean
            specify whether to print the name of dropped column or not
    Returns
    ----------
        a tuple of newly selected lists of numeric and binary columns
        of most important features
    """

    worst_columns = set()

    X_train_new = X_train.copy()
    X_val_new = X_val.copy()

    for i in range(n_feats_to_drop):

        score = 1
        worst_column = None

        for column in X_train_new.columns:
            X_train_dropped = X_train_new.drop(columns=[column])
            X_val_dropped = X_val_new.drop(columns=[column])

            new_cols = set(X_train_dropped.columns)

            col_transformer = make_column_transformer(
                (StandardScaler(), list(set(numeric_features).intersection(new_cols))),
                (
                    OneHotEncoder(drop="if_binary", dtype="int"),
                    list(set(binary_features).intersection(new_cols))
                )
            )

            pipe = make_pipeline(
                col_transformer, DecisionTreeClassifier()
            )

            results = cross_validate_model(
                pipe,
                X_train_dropped,
                y_train,
                X_val_dropped,
                y_val,
                scoring=scoring_metrics
            )

            if results["test_accuracy"] < score:
                score = results["test_accuracy"]
                worst_column = column

        worst_columns.add(worst_column)
        print(f"Dropped {worst_column}")

        X_train_new = X_train.drop(columns=list(worst_columns))
        X_val_new = X_val.drop(columns=list(worst_columns))

    new_cols = X_train_new.columns

    new_numeric_feats = list(set(numeric_features).intersection(new_cols))
    new_binary_feats = list(set(binary_features).intersection(new_cols))

    assert len(new_cols) == len(X_train.columns) - n_feats_to_drop
    assert len(new_numeric_feats) + len(new_binary_feats) == len(new_cols)

    return new_numeric_feats, new_binary_feats
```

In [50]:
```
n_feats_to_keep = 10

model = DecisionTreeClassifier(class_weight="balanced")

numeric_features, binary_features = select_features(
    model, X_train, y_train, X_val, y_val,
    numeric_features_org, binary_features_org,
    n_feats_to_drop=len(X_train.columns) - n_feats_to_keep,
    verbose=True
)
```

```
Dropped lexical_density
Dropped adjective_counts
Dropped asian_context_feature
Dropped asian_imp_word
```

In [51]:
```
assert len(numeric_features + binary_features) == n_feats_to_keep

print(f"The number of features in the final training set: {n_feats_to_keep}")
```

```
The number of features in the final training set: 10
```

Column Transformations

In [52]:
```
# Column Transformation

column_transformer = make_column_transformer(
    (StandardScaler(), numeric_features),
    (OneHotEncoder(drop="if_binary", dtype="int"), binary_features)
)
```

In [53]:
```
column_transformer.fit(X_train)
```

Out[53]:
```
ColumnTransformer(transformers=[('standardscaler', StandardScaler(),
                                 ['average_sentence_length', 'reading_ease',
                                  'punctuation_count', 'noun_counts',
                                  'oov_word_counts', 'verb_counts',
                                  'type_token_ratio', 'average_word_length',
                                  'text_length']),
                                ('onehotencoder',
                                 OneHotEncoder(drop='if_binary', dtype='int'),
                                 ['non_asian_imp_word'])])
```

Training Begins

Dummy Classifier

In [54]:
```
results["dummy_classifier"] = cross_validate_model(
    DummyClassifier(),
    X_train,
    y_train,
    X_val,
    y_val,
    scoring=scoring_metrics
)
```

```
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/miniconda3/envs/573/lib/python3.9/site-packages/sklearn/metrics/_classification.py:13
```

```
08: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predic
ted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [55]:

```
pd.DataFrame(results)
```

Out[55]:

	dummy_classifier
train_accuracy	0.664872
test_accuracy	0.589431
train_precision	0.000000
test_precision	0.000000
train_recall	0.000000
test_recall	0.000000
train_f1	0.000000
test_f1	0.000000

Logistic Regression With Polynomial Features

In [56]:

```
pipe_lr_poly = make_pipeline(
    column_transformer,
    PolynomialFeatures(degree=2),
    LogisticRegression(
        class_weight="balanced",
        max_iter=10000,
        n_jobs=-1,
        random_state=42
    )
)

results["lr_poly"] = cross_validate_model(
    pipe_lr_poly,
    X_train,
    y_train,
    X_val,
    y_val,
    scoring=scoring_metrics
)
```

In [57]:

```
pd.DataFrame(results)
```

Out[57]:

	dummy_classifier	lr_poly
train_accuracy	0.664872	0.679677
test_accuracy	0.589431	0.634146
train_precision	0.000000	0.514986
test_precision	0.000000	0.542636
train_recall	0.000000	0.759036
test_recall	0.000000	0.693069
train_f1	0.000000	0.613636
test_f1	0.000000	0.608696

In [58]:

```
pipe_lr_poly.fit(X_train, y_train)
```

Out[58]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('standardscaler',
                                                  StandardScaler(),
                                                  ['average_sentence_length',
                                                   'reading_ease',
                                                   'punctuation_count',
                                                   'noun_counts',
                                                   'oov_word_counts',
                                                   'verb_counts',
                                                   'type_token_ratio',
                                                   'average_word_length',
                                                   'text_length']),
                                                 ('onehotencoder',
                                                  OneHotEncoder(drop='if_binary',
                                                                dtype='int'),
                                                  ['non_asian_imp_word'])])),
                ('polynomialfeatures', PolynomialFeatures()),
                ('logisticregression',
                 LogisticRegression(class_weight='balanced', max_iter=10000,
                                    n_jobs=-1, random_state=42))])
```

In [59]:

```
pipe_lr_poly.named_steps['polynomialfeatures'].n_output_features_
```

Out[59]:

```
66
```

Logistic Regression With Polynomial Features And Feature Elimination

In [60]:

```
rfe = RFE(Ridge(), n_features_to_select=30)

pipe_lr_poly_rfe = make_pipeline(
    column_transformer,
    PolynomialFeatures(degree=2),
    rfe,
    LogisticRegression(
        class_weight="balanced",
        max_iter=10000,
        n_jobs=-1,
        random_state=42
    )
)

results["lr_poly_rfe"] = cross_validate_model(
    pipe_lr_poly_rfe,
    X_train,
    y_train,
    X_val,
    y_val,
    return_train_score=True,
    scoring=scoring_metrics
)
```

In [61]:

```
pd.DataFrame(results)
```

Out[61]:

	dummy_classifier	lr_poly	lr_poly_rfe
train_accuracy	0.664872	0.679677	0.675639
test_accuracy	0.589431	0.634146	0.642276

	dummy_classifier	lr_poly	lr_poly_rfe
train_precision	0.000000	0.514986	0.511173
test_precision	0.000000	0.542636	0.551181
train_recall	0.000000	0.759036	0.734940
test_recall	0.000000	0.693069	0.693069
train_f1	0.000000	0.613636	0.602965
test_f1	0.000000	0.608696	0.614035

Decision Tree

In [62]:

```
pipe_decision_tree = make_pipeline(
    column_transformer, DecisionTreeClassifier(max_depth=3, class_weight="balanced")
)

results["decision_tree_classifier"] = cross_validate_model(
    pipe_decision_tree,
    X_train,
    y_train,
    X_val,
    y_val,
    return_train_score=True,
    scoring=scoring_metrics
)
```

In [63]:

```
pd.DataFrame(results)
```

Out[63]:

	dummy_classifier	lr_poly	lr_poly_rfe	decision_tree_classifier
train_accuracy	0.664872	0.679677	0.675639	0.679677
test_accuracy	0.589431	0.634146	0.642276	0.601626
train_precision	0.000000	0.514986	0.511173	0.516717
test_precision	0.000000	0.542636	0.551181	0.513514
train_recall	0.000000	0.759036	0.734940	0.682731
test_recall	0.000000	0.693069	0.693069	0.564356
train_f1	0.000000	0.613636	0.602965	0.588235
test_f1	0.000000	0.608696	0.614035	0.537736

Random Forest

In [64]:

```
pipe_rf = make_pipeline(
    column_transformer, RandomForestClassifier(class_weight="balanced")
)

results["rf_classifier"] = cross_validate_model(
    pipe_rf,
    X_train,
    y_train,
    X_val,
    y_val,
    return_train_score=True,
```

```
    scoring=scoring_metrics
)
```

In [65]:

```
pd.DataFrame(results)
```

Out[65]:

	dummy_classifier	lr_poly	lr_poly_rfe	decision_tree_classifier	rf_classifier
train_accuracy	0.664872	0.679677	0.675639	0.679677	1.000000
test_accuracy	0.589431	0.634146	0.642276	0.601626	0.617886
train_precision	0.000000	0.514986	0.511173	0.516717	1.000000
test_precision	0.000000	0.542636	0.551181	0.513514	0.566038
train_recall	0.000000	0.759036	0.734940	0.682731	1.000000
test_recall	0.000000	0.693069	0.693069	0.564356	0.297030
train_f1	0.000000	0.613636	0.602965	0.588235	1.000000
test_f1	0.000000	0.608696	0.614035	0.537736	0.389610

LGBM

In [66]:

```
pipe_lgbm = make_pipeline(
    column_transformer, LGBMClassifier(class_weight="balanced")
)

results["lgbm_classifier"] = cross_validate_model(
    pipe_lgbm,
    X_train,
    y_train,
    X_val,
    y_val,
    return_train_score=True,
    scoring=scoring_metrics
)
```

In [67]:

```
pd.DataFrame(results)
```

Out[67]:

	dummy_classifier	lr_poly	lr_poly_rfe	decision_tree_classifier	rf_classifier	lgbm_classifier
train_accuracy	0.664872	0.679677	0.675639	0.679677	1.000000	0.995962
test_accuracy	0.589431	0.634146	0.642276	0.601626	0.617886	0.621951
train_precision	0.000000	0.514986	0.511173	0.516717	1.000000	0.992000
test_precision	0.000000	0.542636	0.551181	0.513514	0.566038	0.554054
train_recall	0.000000	0.759036	0.734940	0.682731	1.000000	0.995984
test_recall	0.000000	0.693069	0.693069	0.564356	0.297030	0.405941
train_f1	0.000000	0.613636	0.602965	0.588235	1.000000	0.993988
test_f1	0.000000	0.608696	0.614035	0.537736	0.389610	0.468571

XGBoost

In [68]:

```
pipe_xgb = make_pipeline(
    column_transformer, XGBClassifier()
```

```
)

results["xgb_classifier"] = cross_validate_model(
    pipe_xgb,
    X_train,
    y_train,
    X_val,
    y_val,
    return_train_score=True,
    scoring=scoring_metrics
)
```

```
[13:59:28] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1614825350330/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
/opt/miniconda3/envs/573/lib/python3.9/site-packages/xgboost/sklearn.py:888: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

In [69]:

```
pd.DataFrame(results)
```

Out[69]:

	dummy_classifier	lr_poly	lr_poly_rfe	decision_tree_classifier	rf_classifier	lgbm_classifier
train_accuracy	0.664872	0.679677	0.675639	0.679677	1.000000	0.995962
test_accuracy	0.589431	0.634146	0.642276	0.601626	0.617886	0.621951
train_precision	0.000000	0.514986	0.511173	0.516717	1.000000	0.992000
test_precision	0.000000	0.542636	0.551181	0.513514	0.566038	0.554054
train_recall	0.000000	0.759036	0.734940	0.682731	1.000000	0.995984
test_recall	0.000000	0.693069	0.693069	0.564356	0.297030	0.405941
train_f1	0.000000	0.613636	0.602965	0.588235	1.000000	0.993988
test_f1	0.000000	0.608696	0.614035	0.537736	0.389610	0.468571

CatBoost

In [70]:

```
pipe_cat_boost = make_pipeline(
    column_transformer, CatBoostClassifier(
        auto_class_weights="Balanced",
        verbose=False
    )
)

results["cat_boost_classifier"] = cross_validate_model(
    pipe_cat_boost,
    X_train,
    y_train,
    X_val,
    y_val,
    return_train_score=True,
    scoring=scoring_metrics
)
```

In [71]:

```
pd.DataFrame(results)
```

Out[71]:

	dummy_classifier	lr_poly	lr_poly_rfe	decision_tree_classifier	rf_classifier	lgbm_classifier
train_accuracy	0.664872	0.679677	0.675639	0.679677	1.000000	0.995962
test_accuracy	0.589431	0.634146	0.642276	0.601626	0.617886	0.621951
train_precision	0.000000	0.514986	0.511173	0.516717	1.000000	0.992000
test_precision	0.000000	0.542636	0.551181	0.513514	0.566038	0.554054
train_recall	0.000000	0.759036	0.734940	0.682731	1.000000	0.995984
test_recall	0.000000	0.693069	0.693069	0.564356	0.297030	0.405941
train_f1	0.000000	0.613636	0.602965	0.588235	1.000000	0.993988
test_f1	0.000000	0.608696	0.614035	0.537736	0.389610	0.468571

In [ ]:

Feature Importances

Imports

In [72]:

```
import shap
```

In [ ]:

In [73]:

```
column_transformer_shap = make_column_transformer(
    (StandardScaler(), numeric_features_org),
    (OneHotEncoder(drop="if_binary", dtype="int"), binary_features_org)
)
```

In [74]:

```
column_transformer_shap.fit(X_train, y_train)
```

Out[74]:

```
ColumnTransformer(transformers=[('standardscaler', StandardScaler(),
                                 ['average_sentence_length', 'reading_ease',
                                  'adjective_counts', 'punctuation_count',
                                  'noun_counts', 'oov_word_counts',
                                  'verb_counts', 'type_token_ratio',
                                  'average_word_length', 'lexical_density',
                                  'text_length']),
                                ('onehotencoder',
                                 OneHotEncoder(drop='if_binary', dtype='int'),
                                 ['asian_context_feature', 'asian_imp_word',
                                  'non_asian_imp_word'])])
```

In [75]:

```
pipe_xgb = make_pipeline(
    column_transformer_shap, XGBClassifier()
)
```

In [76]:

```
standard_scaler_names = pipe_xgb.named_steps['columntransformer'].named_transformers_['sta
ohe_names = pipe_xgb.named_steps['columntransformer'].named_transformers_['onehotencoder']

all_feature_names = standard_scaler_names + ohe_names
```

In [77]:
```
all_feature_names
```

Out[77]:
```
['average_sentence_length',
 'reading_ease',
 'adjective_counts',
 'punctuation_count',
 'noun_counts',
 'oov_word_counts',
 'verb_counts',
 'type_token_ratio',
 'average_word_length',
 'lexical_density',
 'text_length',
 'asian_context_feature_1',
 'asian_imp_word_1',
 'non_asian_imp_word_1']
```

Feature Importances For Individual Training Examples

In [78]:
```
X_train_enc = pd.DataFrame(
    data=column_transformer_shap.transform(X_train),
    columns=all_feature_names,
    index=X_train.index,
)
X_train_enc.head()
```

Out[78]:

	average_sentence_length	reading_ease	adjective_counts	punctuation_count	noun_counts	oov_word_cou
0	1.223431	-0.812504	0.207487	0.355641	1.718711	1.612
1	-0.043975	-0.170301	-0.432194	-0.779014	-0.697937	-0.585
2	0.244072	-0.364993	0.527328	0.489129	2.241230	1.172
3	-0.248809	0.105584	-0.325580	0.021919	-0.273390	-0.145
4	-0.166875	0.622071	-0.858648	-0.712270	-0.730594	-0.695

Feature Importances For Individual Test Examples

In [79]:
```
X_test_enc = pd.DataFrame(
    data=column_transformer_shap.transform(X_test),
    columns=all_feature_names,
    index=X_test.index,
)
X_test_enc.head()
```

Out[79]:

	average_sentence_length	reading_ease	adjective_counts	punctuation_count	noun_counts	oov_word_cou
0	-0.573981	0.616653	-0.432194	-0.512036	-0.404020	-0.695
1	1.307925	-1.453681	-0.005740	-0.311803	-0.142761	-0.255
2	-0.402434	0.324074	-0.005740	-0.111570	-0.599964	-0.585
3	1.523000	-1.032929	-0.645421	-0.779014	-0.175418	0.073
4	0.022596	0.204027	0.420714	0.822851	0.510387	0.073

In [80]:
```
pipe_xgb.fit(X_train, y_train)
```

```
[13:59:31] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1614825350330/work/sr
c/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the
```

```
objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_met
ric if you'd like to restore the old behavior.
The use of label encoder in XGBClassifier is deprecated and will be removed in a future re
lease. To remove this warning, do the following: 1) Pass option use_label_encoder=False wh
en constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting w
ith 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

Out[80]:

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('standardscaler',
                                                  StandardScaler(),
                                                  ['average_sentence_length',
                                                   'reading_ease',
                                                   'adjective_counts',
                                                   'punctuation_count',
                                                   'noun_counts',
                                                   'oov_word_counts',
                                                   'verb_counts',
                                                   'type_token_ratio',
                                                   'average_word_length',
                                                   'lexical_density',
                                                   'text_length']),
                                                 ('onehotencoder',
                                                  OneHotEncoder(drop='if_bina...
                               colsample_bytree=1, gamma=0, gpu_id=-1,
                               importance_type='gain',
                               interaction_constraints='',
                               learning_rate=0.300000012, max_delta_step=0,
                               max_depth=6, min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=100,
                               n_jobs=8, num_parallel_tree=1, random_state=0,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                               subsample=1, tree_method='exact',
                               validate_parameters=1, verbosity=None))])
```

In [81]:

```
xgb_explainer = shap.TreeExplainer(pipe_xgb.named_steps["xgbclassifier"])
train_xgb_shap_values = xgb_explainer.shap_values(X_train_enc)
```

In [82]:

```
# We are only extracting shapely values for the first 100 test examples for speed.
test_lgbm_shap_values = xgb_explainer.shap_values(X_test_enc[:100])
```

In [83]:

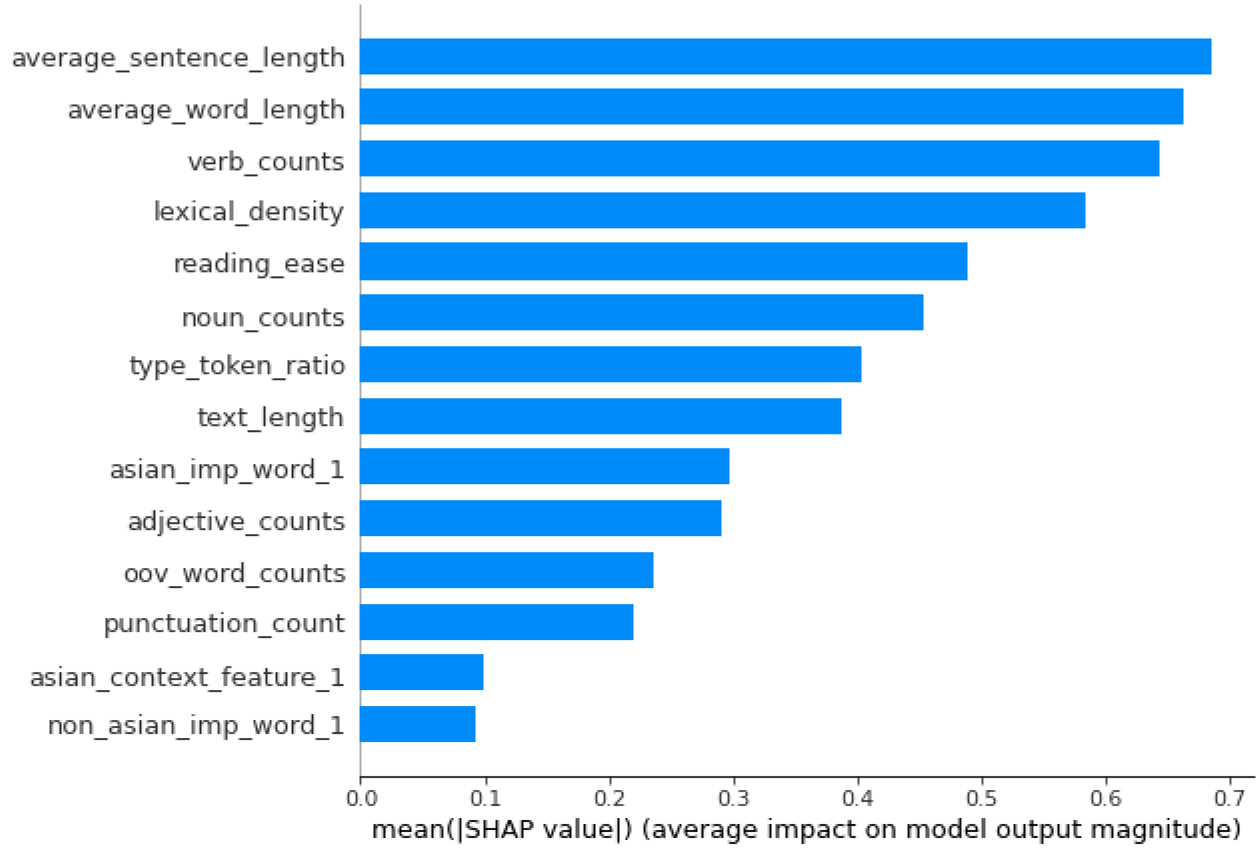
```
shap.initjs()
```

js

Average Feature Importance For Training Examples

In [84]:

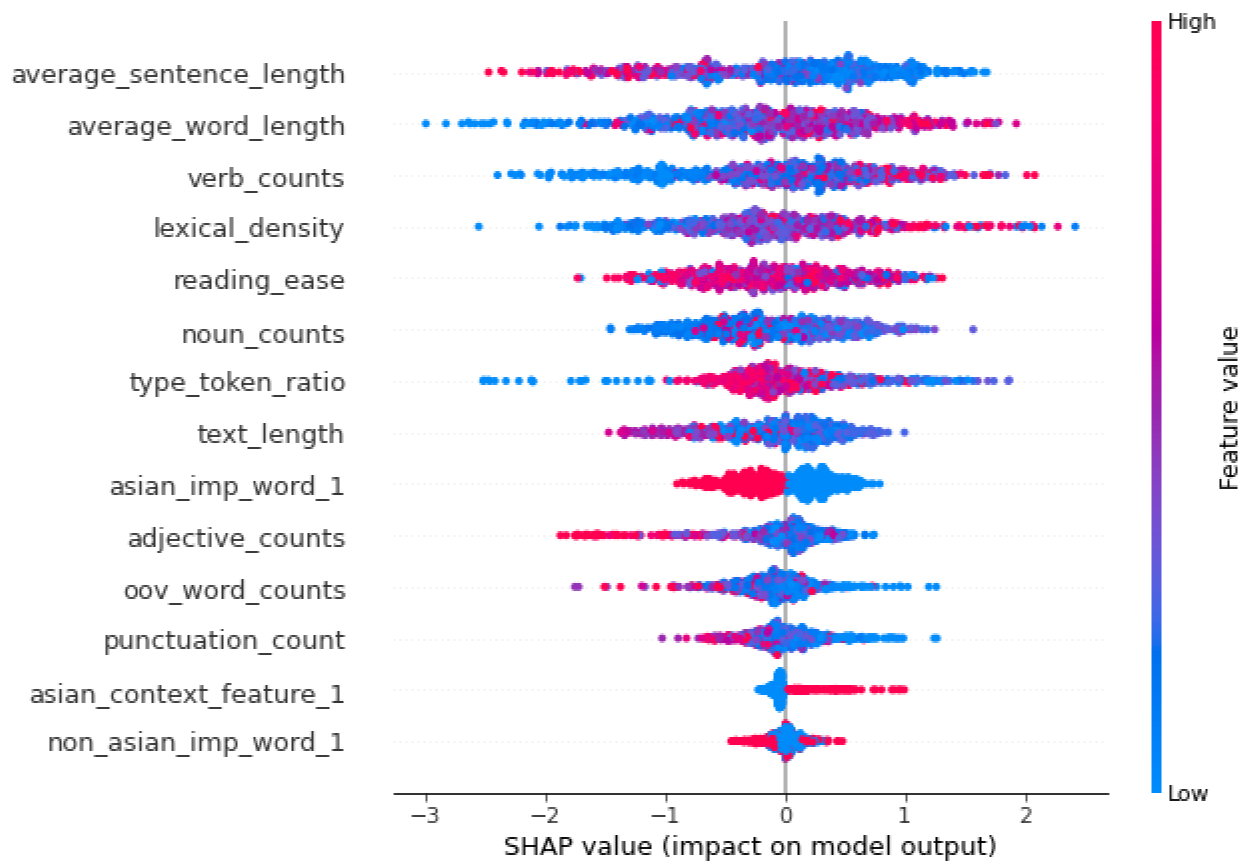
```
shap.summary_plot(train_xgb_shap_values, X_train_enc, plot_type="bar")
plt.savefig('bar.png')
```




```
<Figure size 432x288 with 0 Axes>
```

In [85]:

```
shap.summary_plot(train_xgb_shap_values, X_train_enc)
plt.savefig('summary.png')
```




```
<Figure size 432x288 with 0 Axes>
```

Evaluation on the test set

In [86]:
```
# XGBoost is performing the best

pipe_xgb.fit(X_train, y_train)

score = pipe_xgb.score(X_test, y_test)
```

```
[13:59:32] WARNING: /Users/runner/miniforge3/conda-bld/xgboost-split_1614825350330/work/sr
c/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the
objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_met
ric if you'd like to restore the old behavior.
The use of label encoder in XGBClassifier is deprecated and will be removed in a future re
lease. To remove this warning, do the following: 1) Pass option use_label_encoder=False wh
en constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting w
ith 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

In [87]:
```
print(f"Accuracy on the test set using XGB: {np.round(score, 3)}")
```

```
Accuracy on the test set using XGB: 0.69
```

In [ ]:

Report

Feature creation

Text length (1)

This is a numeric feature that represents the number of total words in a text since there may be a difference in the amount of text that the different L2 English speakers write.

Lexical density (1)

This is a numeric feature which represents the ratio of open class words to total words since the grammar of the European languages is more similar to English, that they would be able to include more of the closed class words.

Average sentence length (1)

This is a numeric feature which represents the average sentence length of a text. It would be interesting to investigate whether there is a difference in sentence length between the two groups of L2 English speakers.

Type token ratio (1)

This is a numeric feature which represents the type to token ratio which essentially is the amount of unique words divided by the total number of words.

Average word length (1)

This is a numeric feature which represents the average sentence length of a text. Since many words are borrowed from French and German to English, this could potentially translate to European L2 speakers using longer words.

Counts of different parts of speech (POS) (3)

This consists of multiple numeric features which represent the number of nouns, verbs and adjectives are in a text.Considered these three open class parts of speech (nouns, verbs and adjectives) because they contained meaningful words.

Number of out of vocabulary words (OOV) words (1)

This is a numeric feature which represents the number of words that are out of the English dictionary, to see if there was a difference between the amount of incorrect words in the text between the two groups.

Reading Ease (1)

This is a numeric feature which represents the reading ease which is calculated using the Flesch–Kincaid formula, to see if there was a difference in readability between the two groups.

(Flesch R (1948). "A new readability yardstick". Journal of Applied Psychology. 32 (3): 221–233. doi:10.1037/h0057532. PMID 18867058.)

Number of punctuations (1)

This is a numeric feature which represent the number of punctuations in the text, since European languages follow similar punctuation rules as English, if they would be different to the Asian L1 speakers.

Asian Context Words (1)

This is a binary feature which is 1 if the text contains a word that is in our Asian context lexicon and 0 otherwise, created this lexicon by manually going through the Asian texts and selecting the most common words that were observed.

Asian Importance Words (1)

This is a binary feature which is 1 if the text contains a word that is in our Asian importance lexicon and 0 otherwise, created this lexicon by going through the Asian texts and selecting the most important words by using TF-IDF.

Non-Asian Importance Words (1)

This is a binary feature which is 1 if the text contains a word that is in our Non-Asian importance lexicon and 0 otherwise, created this lexicon by going through the Non-Asian texts and selecting the most important words by using TF-IDF.

Feature selection and performance

In order to determine which 10 features to choose, performed feature ablation. This involves creating a DecisionTree Classifier and iterating through all the features and drop one at a time to see the impact on the model's performance on the development data set. The features that were decided to drop were `lexical_density`, `type_token_ratio`, `non_asian_imp_word` and `avreage_word_length`. These 4 had the highest scores on the development data when they were removed from the dataset. The remaining and final features were:

- `reading_ease`
- `verb_counts`
- `noun_counts`
- `punctuation_count`
- `adjective_counts`
- `oov_word_counts`
- `average_sentence_length`
- `text_length`
- `asian_context_feature`
- `asian_imp_word`

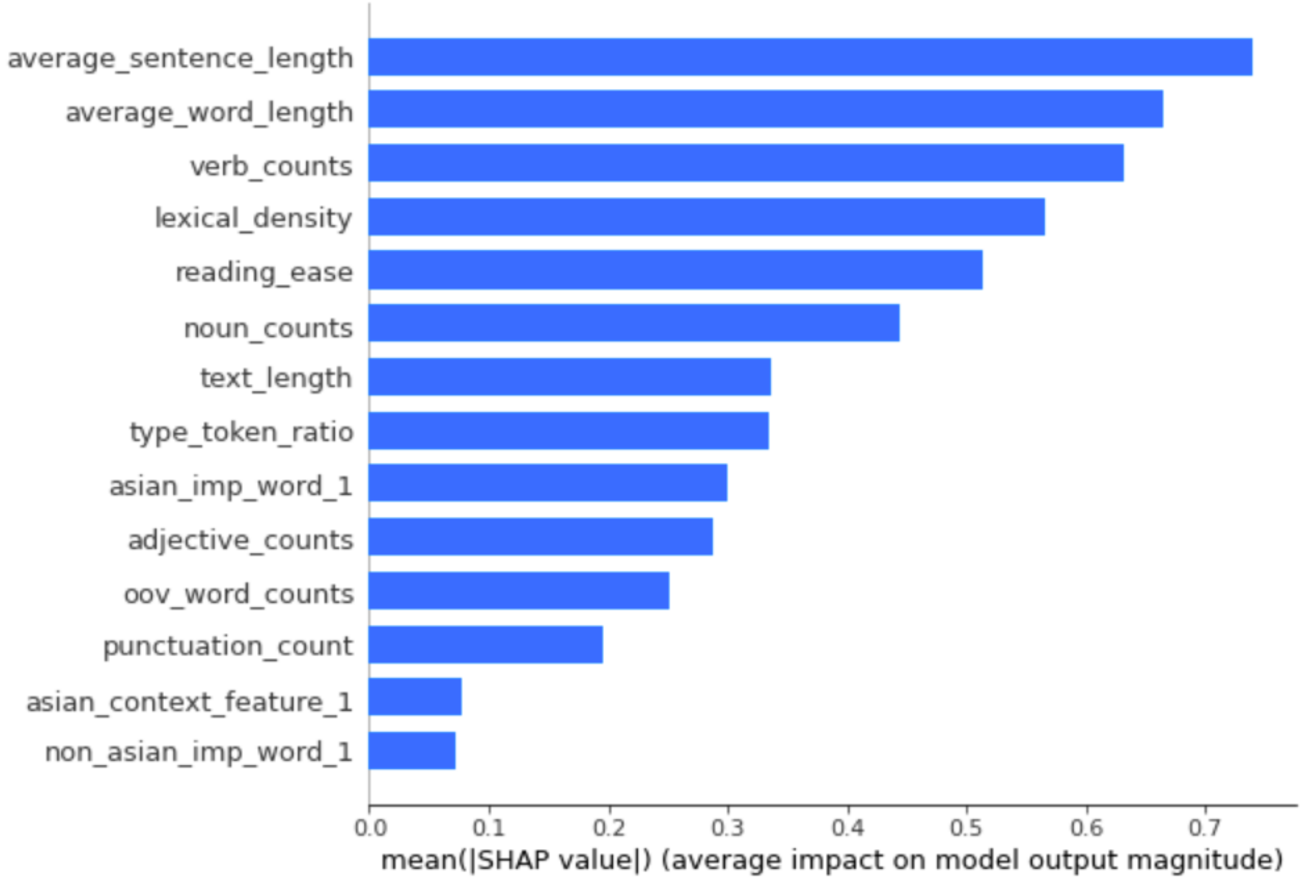
After selection of features, decided to try various different models in order to find something that would best fit our features. Overall, tried 8 models and achieved the following results for the development set. Since there was a class imbalance, also added `class_weight="balanced"` to our model in order to address this and this ameliorated the scores for all the models.

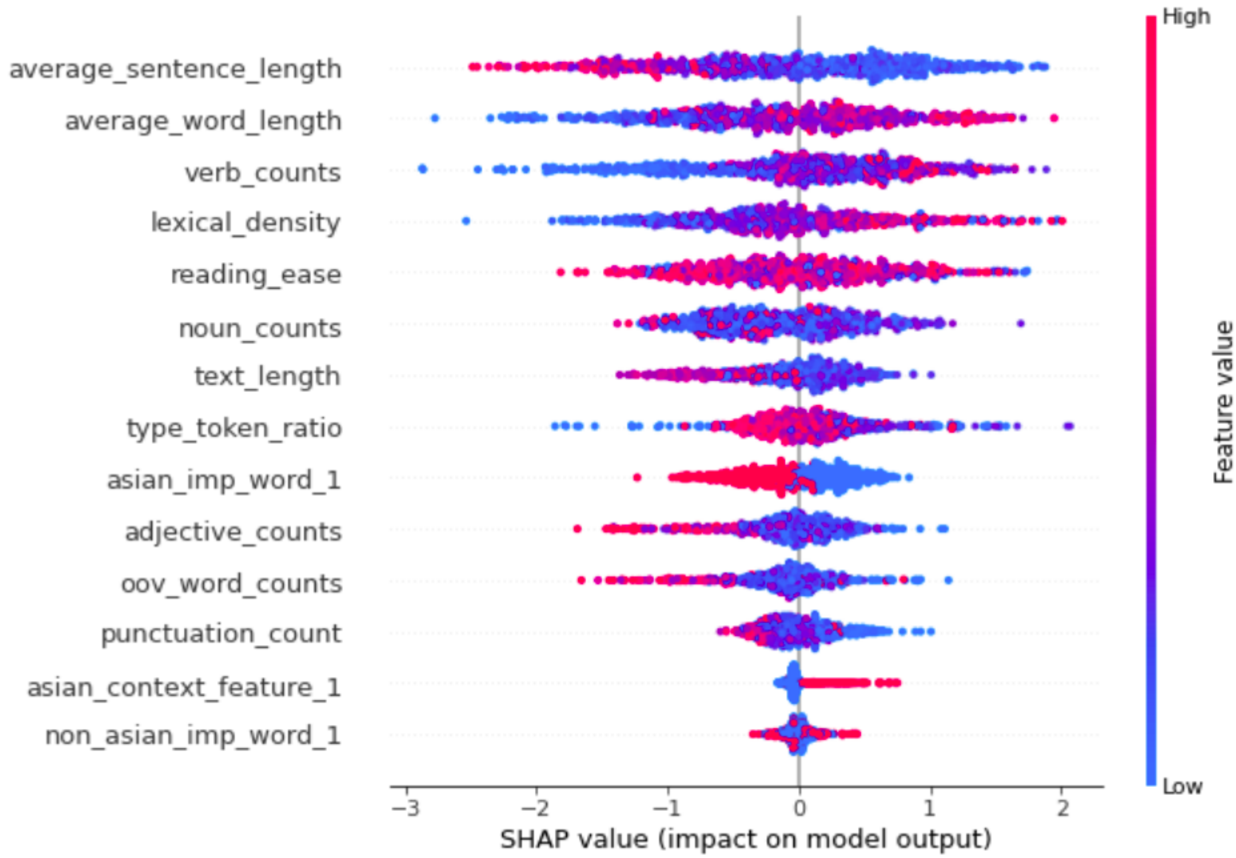
	development accuracy	development precision	development recall	development f1
Dummy Classifier	0.589431	0.0	0.0	0.0
Polynomial Logistic Regression	0.626016	0.536585	0.653465	0.589286
Polynomial Logistic Regression RFE	0.634146	0.541353	0.712871	0.615385
Decision Tree Classifier	0.552846	0.452632	0.425743	0.438776
Random Forest Classifier	0.630081	0.592593	0.316832	0.412903
LGBM Classifier	0.642276	0.576471	0.485149	0.526882
XGB Classifier	0.695122	0.680556	0.485149	0.566474
CatBoost Classifier	0.617886	0.535354	0.524752	0.530000

Found that the XGB Classifier perform the best with our development data.

Thought it would be interesting to investigate within this XGB classifier which features were the most important.

Feature Importance using SHAP (Shapely Additive Explanations)





Summary of Feature Importance

Used SHAP values to understand the contribution of each feature to the prediction. Generated two summary plots as shown above to understand the contribution of the features.

Plot 1

The first plot shows shows the average SHAP value for each feature. Based on the 1st plot generated, we can say that the feature `average_sentence_length` has highest average impact on the model's output (seems to be most important feature) and the feature `non_asian_imp_word` has the lowest average impact on the model's output.

Plot 2

The second plot above shows the most important features for predicting the output and it also shows the direction of how the features are going to drive the prediction. As we can see above, this plot shows the relationship between the value of a feature and the impact on the prediction. Based on the plot above, we can see that few of the features mentioned seem to be important and have significant impact on the predictions :

In the case of `average_sentence_length` , we can see that high feature value would lead to a low SHAP value which means that bigger values for this feature are going to push the predictions to class 0.

In the case of `average_word_length` , we can see that high value of this feature value leads to big SHAP values which means that high values of this feature are going to push the predictions to class 1 and as the value of this feature reduces, would give lower SHAP values and push the predictions towards class 0.

We can see according to the above plots that the top 5 most important features include `average_sentence_length` , `average_word_length` , `lexical_density` , `verb_counts` and `reading_ease` .

We can also observe that the set of important features shown by SHAP are quite different from those selected by the `DecisionTreeClassifier` . Exploring further, we noticed that the scores during feature elimination using Decision Tree were very close and even the features eliminated changed with each execution. There is some scope for improvement here, which we will leave for the next time.

After choosing this model, we decided to perform the final test on our test data. These are the results we achieved

Conclusions

Conclusions

After the series of analysis, we conclude that the XGB Classifier is our best model with the following features:

- `reading_ease`
- `verb_counts`
- `noun_counts`
- `punctuation_count`
- `adjective_counts`
- `oov_word_counts`
- `average_sentence_length`
- `text_length`
- `asian_context_feature`
- `asian_imp_word`

We were able to achieve an accuracy of 0.62

```
In [ ]:
```