ДОДАТОК А

Програмний код

```
class Errorifier():
    def __init__(self):
        self.space_handler = SpaceHandler()
        self.morph = pymorphy2.MorphAnalyzer(lang='uk')

    def tokenize_sentence(self, sentence):
        tokens = re.findall(r'\s+|[^\w\s]|[\w]+', sentence)
        return tokens

    def errorify_sentence(self, word):
        pass

    def errorify_word(self, word):
        pass

    def errorify_dataset(self, dataset: list):
        """input is list of correct sentences, output is list of pairs [(incorrect version, correct version)]"""
        output_dataset = []
        for sentence in tqdm(dataset, desc="Sentence errorification", unit="sentence"):
            output_dataset.append((self.errorify_sentence(sentence), sentence))

        return output_dataset

class SpaceHandler(object):
    """
```

```
    handles spaces before and after punctuation
    functions:
    - space_stripper - strips extra spaces from text, used in space_oddity
    - space_oddity - adds extra spaces before punctuation for tokenization
    """
    def __init__(self):
        self.us = "[A-ЩЬЮЯЄҐІЇЭЫЪa-щьюяєгіїэыъ0-9a-zA-Z()%‰\'№\+]"
        self.upr = r'[.?!,;:—-]' # ukrainian punctuation
        self.uwr = re.compile(self.us + "+")

    def space_stripper(self, sentence): # to get rid of extra spaces
        sentence = re.sub(r"\s{2,}", ' ', sentence) # double+ spaces
        sentence = re.sub(r"^\s+", '', sentence) # a space in the beginning (if double, then has already been removed)
        sentence = re.sub(r"\s+$", '', sentence) # a space in the end
        sentence = re.sub(r'([0-9])([.?!,;:—-])\s([0-9])', r"\1\2\3", sentence) # spaces in punctuation between numbers
        return sentence

    def space_oddity(self, sentence): # to add spaces in between of punctuation
        sentence = self.space_stripper(sentence) # get rid of extra spaces
        words = re.findall(self.uwr, sentence) # match words
        punctuation = re.split(self.uwr, sentence) # split the remains over words. The punctuation will be both at the beginning and in the end
        i = 0 # the index of considered punctuation
        sentence = "" # dummy for the newly created sentence
        while i < len(punctuation) - 1: # end before the last punctuation
            sentence += ''.join(list(punctuation[i])) + ' ' +  words[i] + ' ' # the symbols between words now get to be joined by spaces. Likely with several spaces if there were spaces
```

```
            i += 1
        sentence += ' '.join(list(punctuation[-1])) # add the last punctuation to
account for them not having the word following
        return self.space_stripper(sentence) # strip the remaining spaces just in case

class PunctuationErrorifier(Errorifier):

    def __init__(self):
        super().__init__()
        self.generate_substitution_matrix()

    def generate_substitution_matrix(self):
        self.marks = [' ', ',', ';', ':', chr(8212), '-', '.', '?', '!', chr(8230)]
        self.substitution_matrix = pd.DataFrame(data = np.zeros((len(self.marks),
len(self.marks))),
                                                index = self.marks,
                                                columns = self.marks)
        self.substitution_matrix.loc[:, " "] = 0.8

        # probabilties whitespace to smth

        self.substitution_matrix.loc[' ', ','] = 10 # extra comma
        self.substitution_matrix.loc[' ', ';'] = 1 # extra semicolon
        self.substitution_matrix.loc[' ', ':'] = 1 # extra colon
        self.substitution_matrix.loc[' ', chr(8212)] = 2 # extra dash
        self.substitution_matrix.loc[' ', '-'] = 1 # extra hyphen

        self.substitution_matrix.loc[',', ','] = 30 # kept comma
        self.substitution_matrix.loc[',', ' '] = 80 # missed comma
        self.substitution_matrix.loc[',', ';'] = 1 # comma by semicolon
```

```
        self.substitution_matrix.loc[';', ' '] = 5 # missed semicolon
        self.substitution_matrix.loc[';', ','] = 20 # semicolon by comma
        self.substitution_matrix.loc[';', chr(8212)] = 1 # semicolon by dash

        self.substitution_matrix.loc[':', ' '] = 1 # missed colon
        self.substitution_matrix.loc[':', ','] = 3 # colon by comma
        self.substitution_matrix.loc[':', chr(8212)] = 30 # colon by dash

        self.substitution_matrix.loc[chr(8212), ' '] = 90 # missed dash
        self.substitution_matrix.loc[chr(8212), ','] = 30 # dash by comma
        self.substitution_matrix.loc[chr(8212), ';'] = 1 # dash by semicolon
        self.substitution_matrix.loc[chr(8212), ':'] = 5 # dash by colon

        self.substitution_matrix.loc['-', ' '] = 6 # missed hyphen
        self.substitution_matrix.loc['-', chr(8212)] = 1 # hyphen by dash (uniting
dash)

        self.substitution_matrix.loc[chr(8230), chr(8230)] = 1 # unchanged ellipsis

        self.substitution_matrix.loc['.', '?'] =  1 # into question
        self.substitution_matrix.loc['.', '!'] =  1 # into assertion
        self.substitution_matrix.loc['?', '.'] =  1 # no question
        self.substitution_matrix.loc['?', '!'] =  1 # question into assertion
        self.substitution_matrix.loc['!', '.'] =  1 # no assertion
        self.substitution_matrix.loc['!', '?'] =  1 # assertion into question

        probability_of_error = 5/10 # p of error on a spot

        for i in range(len(self.marks)-2):
```

```
            self.substitution_matrix.iloc[i,:] =
probability_of_error*self.substitution_matrix.iloc[i,:]/np.sum(self.substitution_
matrix.iloc[i,:])
            self.substitution_matrix.iloc[i,i] = 1 - probability_of_error

        # setting up custom probabilities for deleting the punctuation
        to_spaces = .8
        self.substitution_matrix.loc[:,' '] = to_spaces
        norm = np.sum(self.substitution_matrix.iloc[:,1:], axis=1)
        for m in self.marks[1:]:
            self.substitution_matrix.loc[:,m]=(1-
to_spaces)*self.substitution_matrix.loc[:,m]/norm

        no_change_if_space_prob = 0.95
        self.substitution_matrix.loc[' ',' '] = no_change_if_space_prob
        norm = np.sum(self.substitution_matrix.iloc[0:1,1:], axis=1)
        multiplier = (1-no_change_if_space_prob)/float(norm)
        for m in self.marks[1:]:
            self.substitution_matrix.loc[' ',m] = self.substitution_matrix.loc['
',m]*multiplier

        def update_cell(char_from, char_to, prob):
            # set new prob
            self.substitution_matrix.loc[char_from, char_to] = prob
            # normalize row
            norm = np.sum(self.substitution_matrix.loc[char_from,
self.substitution_matrix.columns != char_to])
            multiplier = (1-prob)/float(norm)
```

```
            self.substitution_matrix.loc[char_from, self.substitution_matrix.columns
!= char_to] = self.substitution_matrix.loc[char_from,
self.substitution_matrix.columns != char_to]*multiplier

        update_cell(' ', ' ', 0.95) #to reduce number of deletes
        update_cell(',', ' ', 0.8) #to reduce number of append_,
        update_cell('.', ' ', 0.1) # to reduce number of append_.
        update_cell('.', '?', 0.00005) #to reduce number of replace_.
        update_cell('.', '!', 0.00005) #to reduce number of replace_.
        update_cell('—', ' ', 0.90) #to increase number of append_-
        update_cell(':', ' ', 0.90) #to increase number of append_:
        update_cell('—', ',', 0.05) #to increase number of replace_-
        update_cell(':', ',', 0.05) #to increase number of replace_:

        return

    def tokenize_sentence(self, sentence):
        tokens = re.findall(r'\s+|[^\w\s]|[\w]+', sentence)
        return tokens

    def errorify_sentence(self, sentence):
        sentence = self.space_handler.space_oddity(sentence)
        tokens = self.tokenize_sentence(sentence)

        modified_sentence = []

        for token in tokens:
            if token in self.marks:
                probabilities = self.substitution_matrix.loc[token].values
                new_token = np.random.choice(self.marks, p=probabilities)
```

```
                modified_sentence.append(new_token)
            else:
                modified_sentence.append(token)
        modified_sentence = ''.join(modified_sentence)
        modified_sentence = self.space_oddity(modified_sentence)
        return modified_sentence

class GrammarErrorifier(Errorifier):
    def __init__(self):
        super().__init__()
        self.matchings = {"PROPN":"NOUN","NOUN":"NOUN",
"VERB":"VERB", "PRON":"NPRO", "DET":"NPRO","ADJ":"ADJF",
"NUM":"NUMR"} # match POS for inflector to be readable
        self.p_mispreposition = 0.8
        self.p_miscase = 0.8
        self.p_verb = 0.7
        self.p_adj = 0.8

        self.strategies = {"NOUN": self._process_noun,
                           "VERB": self._process_verb,
                           "ADJF": self._process_adjective}

    def tokenize_sentence(self, sentence):
        tokens = re.findall(r'\s+|[^\w\s]|[\w]+', sentence)
        return tokens

    def _process_noun(self, parsed_word):
        if np.random.random() < self.p_miscase:
            current_case = parsed_word.tag.case
            cases = ["nomn", "gent", "datv", "accs", "ablt", "loct", "voct"]
```

```
            possible_cases = [case for case in cases if case != current_case]
            if possible_cases:
                new_case = np.random.choice(possible_cases)
                inflected_word = parsed_word.inflect({new_case})
                if inflected_word:
                    return inflected_word.word
        return parsed_word.word

    def _process_verb(self, parsed_word):
        if np.random.random() < self.p_verb:
            current_gender = parsed_word.tag.gender
            genders = ["masc", "femn", "neut"]
            possible_genders = [gender for gender in genders if gender !=
current_gender]
            if possible_genders:
                new_gender = np.random.choice(possible_genders)
                inflected_word = parsed_word.inflect({new_gender})
                if inflected_word:
                    return inflected_word.word
        return parsed_word.word

    def _process_adjective(self, parsed_word):
        return self._process_verb(parsed_word)

    def errorify_word(self, word):
        expected = self.morph.parse(word)[0]
        pos = expected.tag.POS

        capitalized = word[0].isupper()
```

```
        if pos in self.strategies:
            result_word = self.strategies[pos](expected)
        else:
            result_word = word

        if capitalized:
            result_word = result_word.capitalize()

        return result_word

    def errorify_sentence(self, sentence):
        sentence = self.space_handler.space_oddity(sentence)
        tokens = self.tokenize_sentence(sentence)
        for i, token in enumerate(tokens):
            if token.isalpha():
                tokens[i] = self.errorify_word(token)
            else:
                pass
        sentence = ''.join(tokens)
        sentence = self.space_handler.space_oddity(sentence)
        return sentence

class TranslitErrorifier(Errorifier):

    def __init__(self, p_errorification=0.8, device=0):
        super().__init__()
        self.p_errorification = p_errorification

        self.translator = pipeline(task="translation", model='Helsinki-NLP/opus-mt-uk-ru', framework='pt', device=device)
```

```
    def transliterate_to_uk(self, word):
        """Transliterate russian word to ukrainian """
        translit_map = {
            "а": "а", "б": "б", "в": "в", "г": "г", "д": "д",
            "е": "є", "ё": "йо", "ж": "ж", "з": "з", "и": "і",
            "й": "й", "к": "к", "л": "л", "м": "м", "н": "н",
            "о": "о", "п": "п", "р": "р", "с": "с", "т": "т",
            "у": "у", "ф": "ф", "х": "х", "ц": "ц", "ч": "ч",
            "ш": "ш", "щ": "щ", "ы": "и", "э": "е", "ю": "ю",
            "я": "я", "ъ": "", "ь": "ь"
        }

        transliterated = "".join(translit_map.get(char, char) for char in word)
        return transliterated

    def apply_grammatical_tags(self, ru_word, expected_uk_word):
        """Get str as ru_word, get Parse() object as expected_uk_word.
        Apply grammatical tags (gender, number, case, etc.) from the Ukrainian word to the Russian word."""

        parsed_word = self.morph.parse(ru_word)[0]

        if expected_uk_word.tag.case and expected_uk_word.tag.case != parsed_word.tag.case:
            inflected_word = parsed_word.inflect({expected_uk_word.tag.case}) or self.morph.parse(ru_word)[0]
        else:
            inflected_word = parsed_word
```

```
        if expected_uk_word.tag.gender and expected_uk_word.tag.gender !=
parsed_word.tag.gender:
            inflected_word =
inflected_word.inflect({expected_uk_word.tag.gender}) or inflected_word

        if expected_uk_word.tag.number and expected_uk_word.tag.number !=
parsed_word.tag.number:
            inflected_word =
inflected_word.inflect({expected_uk_word.tag.number}) or inflected_word

        if expected_uk_word.tag.tense and expected_uk_word.tag.tense !=
parsed_word.tag.tense:
            inflected_word = inflected_word.inflect({expected_uk_word.tag.tense})
or inflected_word

        return inflected_word.word if not isinstance(inflected_word, str) else
inflected_word

    def errorify_word(self, word):
        expected = self.morph.parse(word)[0]
        pos = expected.tag.POS
        capitalized = word[0].isupper()

        if pos in ["NOUN", "VERB"]:
            if np.random.random() < self.p_errorification:
                new_word = self.translator(expected.word)
                new_word = new_word[0].get('translation_text', expected.word)
                new_word = self.apply_grammatical_tags(new_word, expected)
                new_word = self.transliterate_to_uk(new_word)
```

```
            if capitalized:
                new_word = new_word.capitalize()
            return new_word
        return word

    def errorify_sentence(self, sentence):
        sentence = self.space_handler.space_oddity(sentence)
        tokens = self.tokenize_sentence(sentence)
        for i, token in enumerate(tokens):
            if token.isalpha():
                tokens[i] = self.errorify_word(token)
            else:
                pass
        sentence = ''.join(tokens)
        sentence = self.space_handler.space_oddity(sentence)
        return sentence

class AnglicismErrorifier(Errorifier):
    def __init__(self, device=0):
        super().__init__()
        self.p_anglicism_errorification = 0.35
        self.translator_uk_en = pipeline(task="translation", model='Helsinki-
NLP/opus-mt-uk-en', framework='pt', device=device)

    def errorify_word(self, word):
        morph_word = self.morph.parse(word)[0]
        if morph_word.tag.POS in ["NOUN", "VERB"] and np.random.random() <
self.p_anglicism_errorification:
            capitalized = word[0].isupper()
```

```
            result_word = translit(self.translator_uk_en(word)[0]["translation_text"],
"uk")
            if capitalized:
                result_word = result_word.capitalize()
            else:
                result_word = result_word.lower()
            return result_word
        else:
            return word

    def if_errorify_word(self, word):
        """check if script should errorify word, return bool"""
        morph_word = self.morph.parse(word)[0]
        if morph_word.tag.POS in ["NOUN", "VERB"] and np.random.random() <
self.p_anglicism_errorification:
            return True
        return False

    def errorify_sentence(self, sentence):
        sentence = self.space_handler.space_oddity(sentence)
        tokens = self.tokenize_sentence(sentence)
        for i, token in enumerate(tokens):
            if token.isalpha():
                tokens[i] = self.errorify_word(token)
            else:
                pass
        sentence = ''.join(tokens)
        sentence = self.space_handler.space_oddity(sentence)
        return sentence
```

```
    def replace_masks_in_sentences(self, sentences, mask_words):
        mask_index = 0  # Індекс поточного слова з mask_words

        updated_sentences = []

        for sentence in sentences:
            tokens = self.tokenize_sentence(sentence)

            for i, token in enumerate(tokens):
                if token == "MASKHERE123" and mask_index < len(mask_words):
                    tokens[i] = mask_words[mask_index]
                    mask_index += 1  # Переходимо до наступного слова з
mask_words

            # Об'єднуємо слова назад в речення
            new_sentence = " ".join(tokens)
            new_sentence = self.space_handler.space_oddity(new_sentence)
            updated_sentences.append(new_sentence)

        return updated_sentences

    def transliterate_list_of_words(self, words):
        """list of dict with {"generated_text": 'text'}"""
        result = []
        for word in words:
            result_word = translit(word["translation_text"], "uk")
            result.append(result_word)

        return result
```

```
    def errorify_dataset(self, dataset):
        print("WE were here forever")
        words_to_translate = []
        new_dataset = []
        for sentence in dataset:
            sentence = self.space_handler.space_oddity(sentence)
            tokens = self.tokenize_sentence(sentence)
            for i, token in enumerate(tokens):
                if token.isalpha():
                    if self.if_errorify_word(token):
                        words_to_translate.append(token)
                        tokens[i] = "MASKHERE123"
                else:
                    pass
            new_sentence = ''.join(tokens)
            new_sentence = self.space_handler.space_oddity(new_sentence)
            new_dataset.append(new_sentence)
        translated_words = self.translator_uk_en(words_to_translate,
batch_size=32)
        words = self.transliterate_list_of_words(translated_words)
        result = self.replace_masks_in_sentences(new_dataset, words)
        output_dataset = []
        for incorrect, correct in zip (result, dataset):
            output_dataset.append((incorrect,
                                   self.space_handler.space_oddity(correct)))
        return output_dataset
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	83 с.
	Інші документи	
2.	Презентаційні матеріали	21 плакат

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Пилипенко Р.О.			Розроблення системи виправлення граматичних та синтаксичних помилок в україномовних текстах	Шифр групи	Код напр./спец.
Перевірив		Дейнеко А.О.				ДСм-23-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					