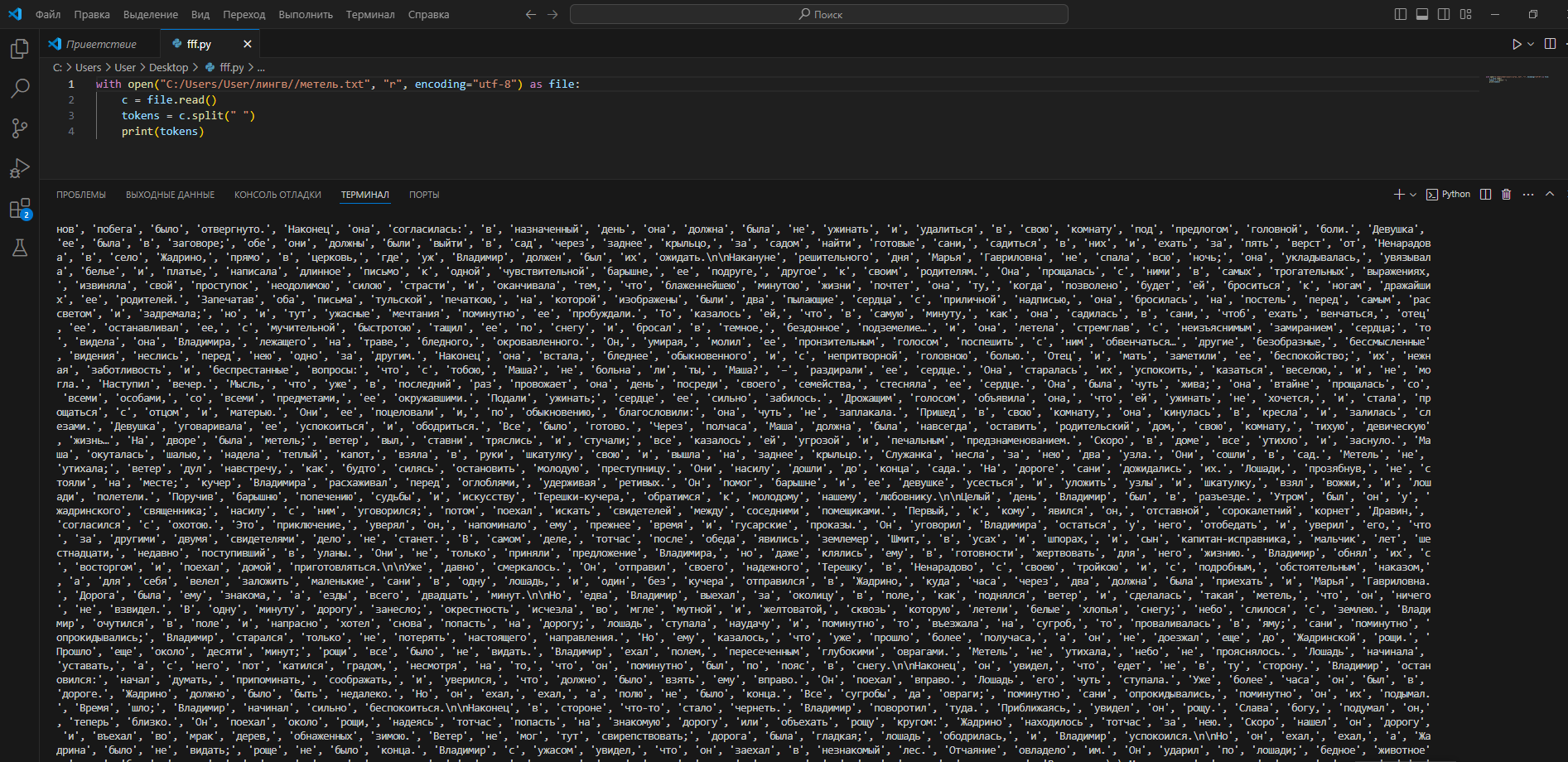
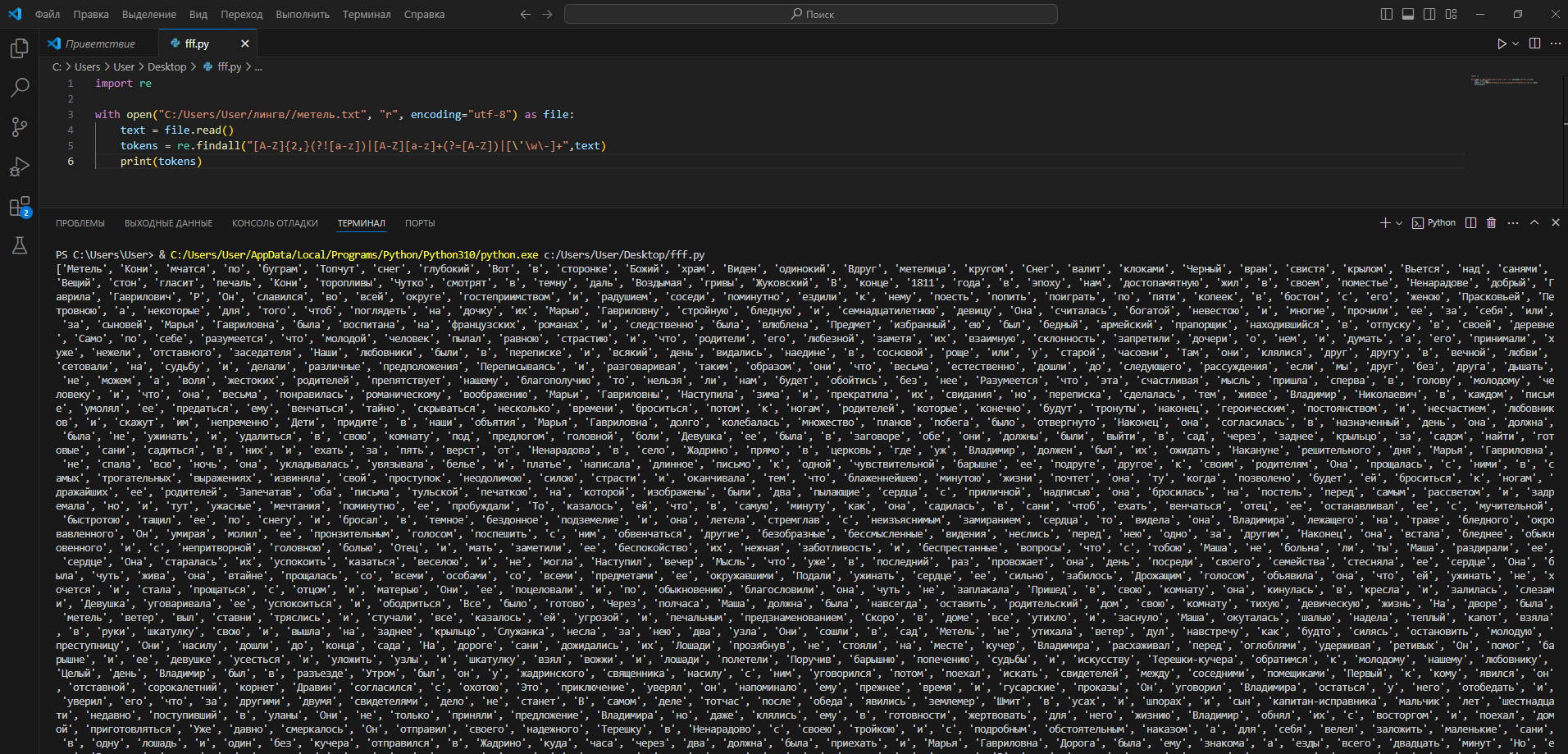
Уровень 1

Проведем токенизацию с помощью split. Берем текст А.С.Пушкина «Метель», пишем программу на Python.



Проведем токенизацию при помощи регулярных выражений



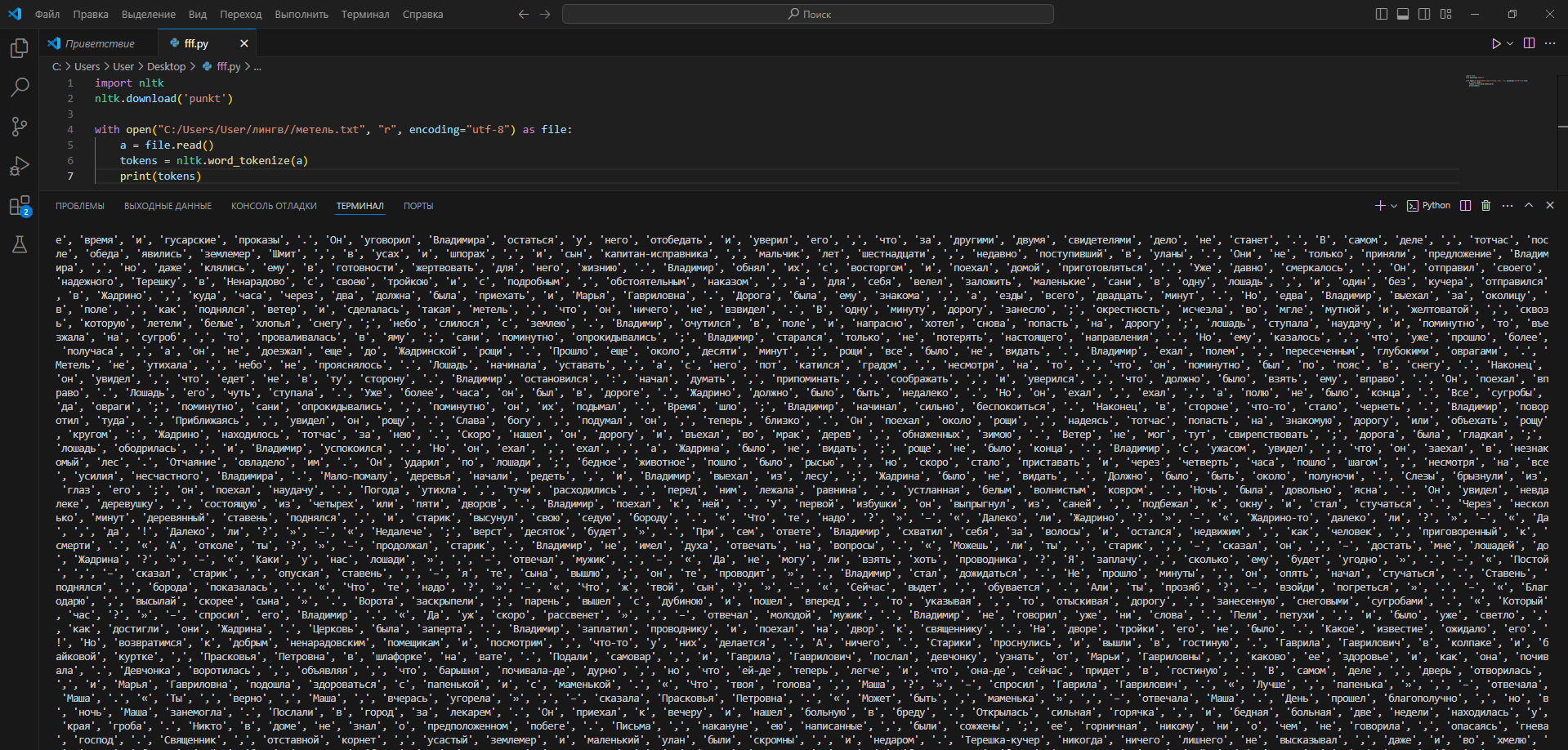
re.findall("[A-Z]{2,}(?![a-z])|[A-Z][a-z]+(?=[A-Z])|[\'\w\-]+", text): Эта строка кода использует модуль re (регулярные выражения) для поиска и извлечения подстрок из текста text, соответствующих шаблонам, заданным в регулярных выражениях.

- [A-Z]{2,}(?![a-z]): Этот шаблон ищет последовательность из двух или более заглавных букв (A-Z), за которыми не следует строчная буква (отрицательный просмотр вперед). Таким образом, это позволяет нам извлекать аббревиатуры, написанные заглавными буквами.

- [A-Z][a-z]+(?=[A-Z]): Этот шаблон ищет одну заглавную букву (A-Z), за которой следует одна или более строчных букв (a-z), перед которыми стоит заглавная буква (положительный просмотр вперед). Этот шаблон помогает нам извлекать слова, начинающиеся с заглавной буквы и содержащие заглавные и строчные буквы.

- [\'\w\-]+: Этот шаблон ищет последовательности символов, которые могут включать в себя апострофы, буквы (как заглавные, так и строчные), цифры и дефисы. Таким образом, он помогает извлекать слова, содержащие указанные символы.

**Проведем токенизацию при помощи библиотеки NLTK**

****

1. import nltk: Эта строка импортирует библиотеку Natural Language Toolkit (NLTK), которая предоставляет множество инструментов для обработки текста на естественном языке.

2. nltk.download('punkt'): Этот код загружает ресурсы, необходимые для работы токенизатора в NLTK. Токенизатор разбивает текст на отдельные слова или токены.

3. with open("C:/Users/User/лингв//метель.txt", "r", encoding="utf-8") as file:: Этот блок открывает файл с именем "метель.txt" для чтения. Файл находится по указанному пути "C:/Users/User/лингв//метель.txt". Он использует кодировку UTF-8 для чтения файла.

4. a = file.read(): Этот код читает содержимое файла и сохраняет его в переменной a как строку.

5. tokens = nltk.word\_tokenize(a): Этот код использует токенизатор из библиотеки NLTK для разделения строки a на отдельные слова (токены) и сохраняет их в переменной tokens.

6. print(tokens): Этот код выводит список токенов (слов) на экран.

### Отличия различных способов токенизации:

1. \*\*Метод .split\*\*:

- Простой в использовании.

- Разделяет текст по заданному разделителю (обычно пробелу).

- Не учитывает пунктуацию и специальные символы.

2. \*\*Токенизация при помощи регулярных выражений\*\*:

- Позволяет более гибко настраивать правила разделения текста.

- Учитывает пунктуацию и специальные символы.

- Может быть сложнее в использовании из-за необходимости работы с регулярными выражениями.

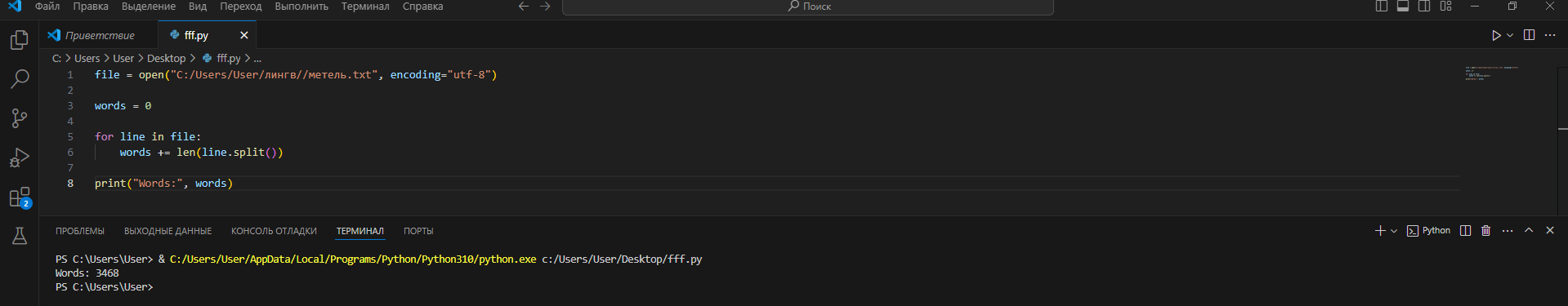
3. \*\*Токенизация при помощи библиотеки NLTK\*\*:

- Предоставляет широкий выбор методов токенизации для различных языков.

- Умеет обрабатывать особенности языка, такие как апострофы, дефисы и другие.

- Может требовать предварительной установки библиотеки.

**Посчитаем количество слов в тексте при помощи len**

****

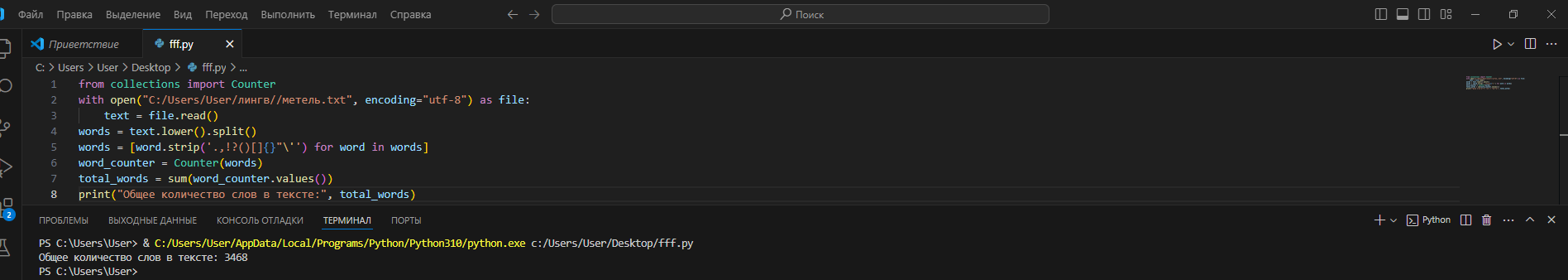
words = 0: Создается переменная words, которая будет использоваться для подсчета количества слов в файле и начальное значение устанавливается как 0.

for line in file:: Этот цикл итерирует по каждой строке в файле.

words += len(line.split()): В этой строке происходит подсчет слов в каждой строке файла. Метод split() разделяет строку на слова (по умолчанию по пробелам) и возвращает список слов. Функция len() используется для подсчета количества слов в списке, после чего это число добавляется к переменной words.

print("Words:", words): После завершения цикла программа выводит общее количество слов в файле.

**Посчитаем количество слов в тексте при помощи Counter**

****

from collections import Counter: Эта строка импортирует класс Counter из модуля collections. Counter используется для подсчета хэшируемых объектов.

text = file.read(): Эта строка читает содержимое файла и сохраняет его в переменной text. Теперь text содержит весь текст из файла.

word\_counter = Counter([word.strip('.,!?()[]{}"\'').lower() for word in text.split()]): В этой строке происходит следующее:

- text.split(): Текст разбивается на слова с помощью метода split(), который разделяет текст по пробелам и создает список слов.

- word.strip('.,!?()[]{}"\'').lower(): Каждое слово в списке обрабатывается следующим образом:

- strip('.,!?()[]{}"\''): Удаляются знаки препинания и другие символы с начала и конца слова.

- lower(): Слово приводится к нижнему регистру.

- [...]: Генератор списка создает новый список слов, очищенных от знаков препинания и в нижнем регистре.

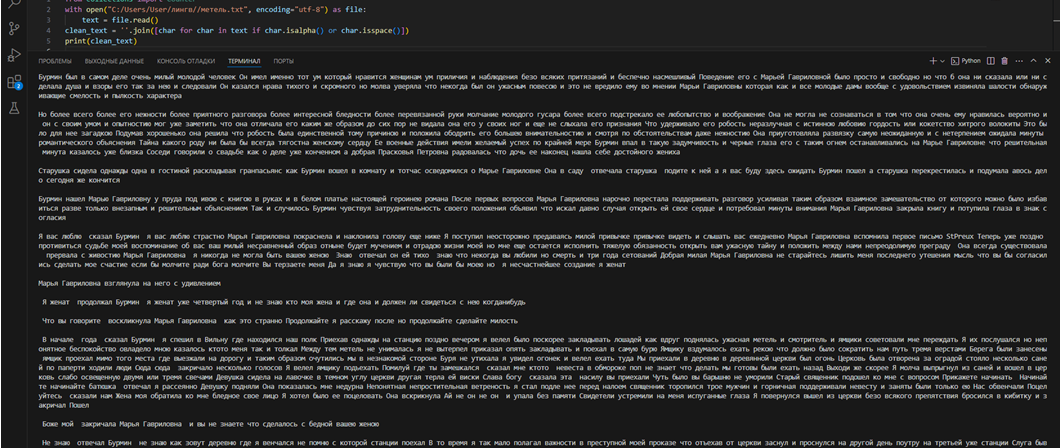
- Counter(...): Создается объект Counter, который подсчитывает количество уникальных слов в списке.

total\_words = sum(word\_counter.values()): В этой строке вычисляется общее количество слов в тексте путем суммирования значений в объекте Counter, который содержит подсчитанное количество каждого слова.

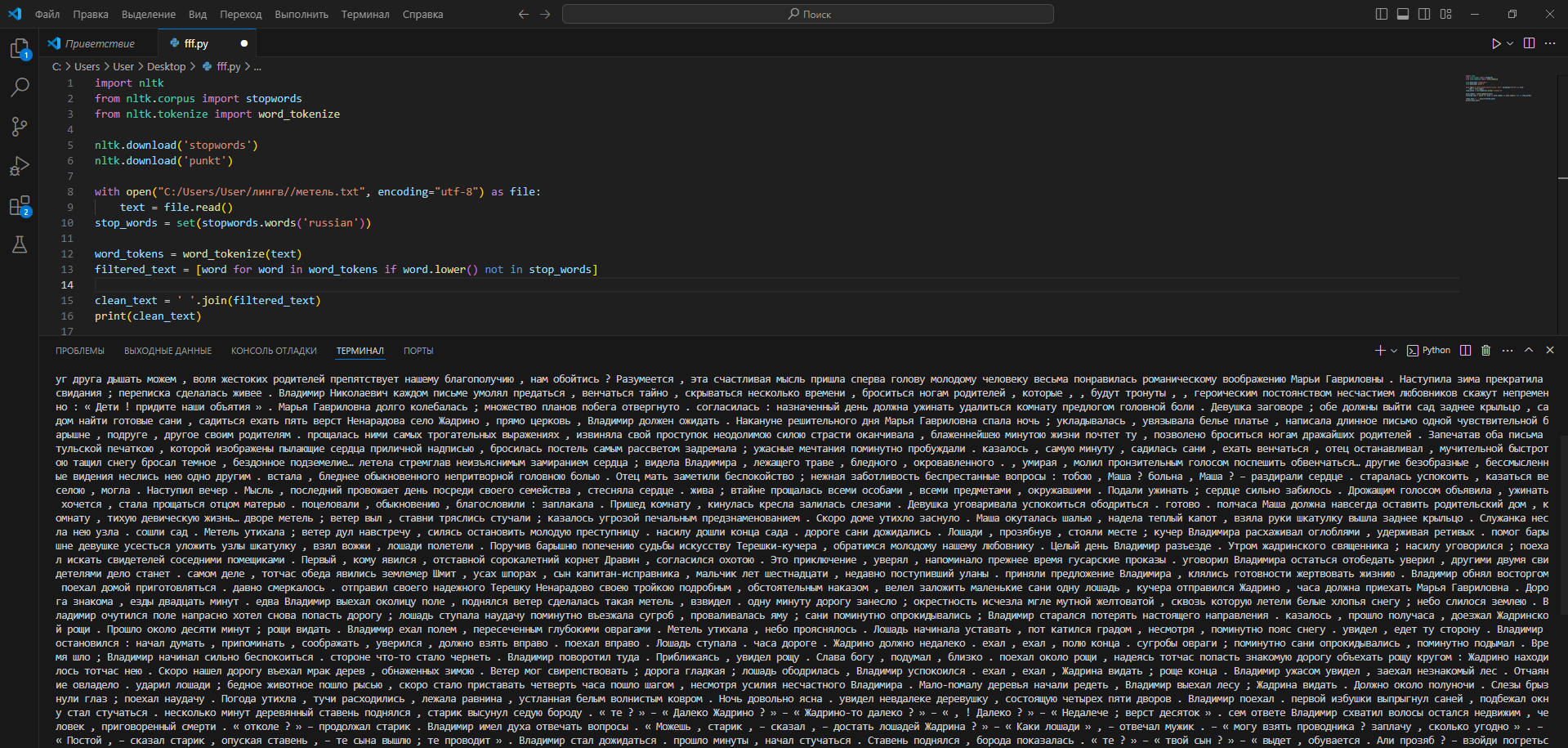
print("Общее количество слов в тексте:", total\_words): Наконец, эта строка выводит общее количество слов в тексте на экран вместе с соответствующим сообщением.

Вывод: разницы в результатах нет

**Очищаем текст от пунктуации при помощи isalpha**

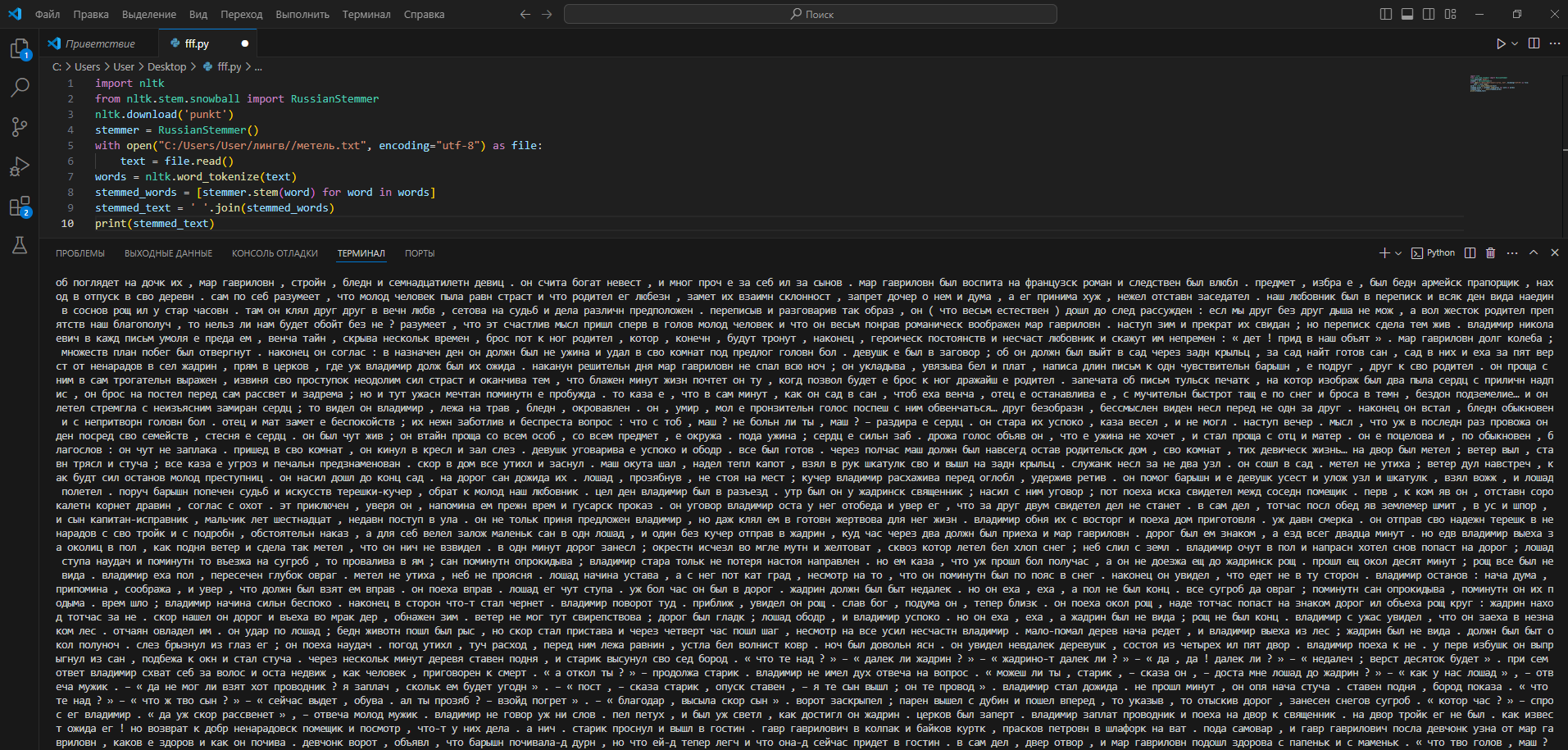
Мы проходим по каждому символу в исходном тексте text и добавляем его к переменной clean\_text, только если символ является буквой (char.isalpha()) или пробелом (char.isspace()). Таким образом, все символы пунктуации удаляются, а оставшиеся символы объединяются в новую строку clean\_text.

**Очищаем текст от стоп-слов при помощи библиотеки NLTK**



В этом коде мы загружаем список стоп-слов для русского языка из библиотеки NLTK, токенизируем исходный текст с помощью word\_tokenize, затем фильтруем токены, оставляя только те, которые не являются стоп-словами. Наконец, объединяем отфильтрованные токены обратно в строку clean\_text.

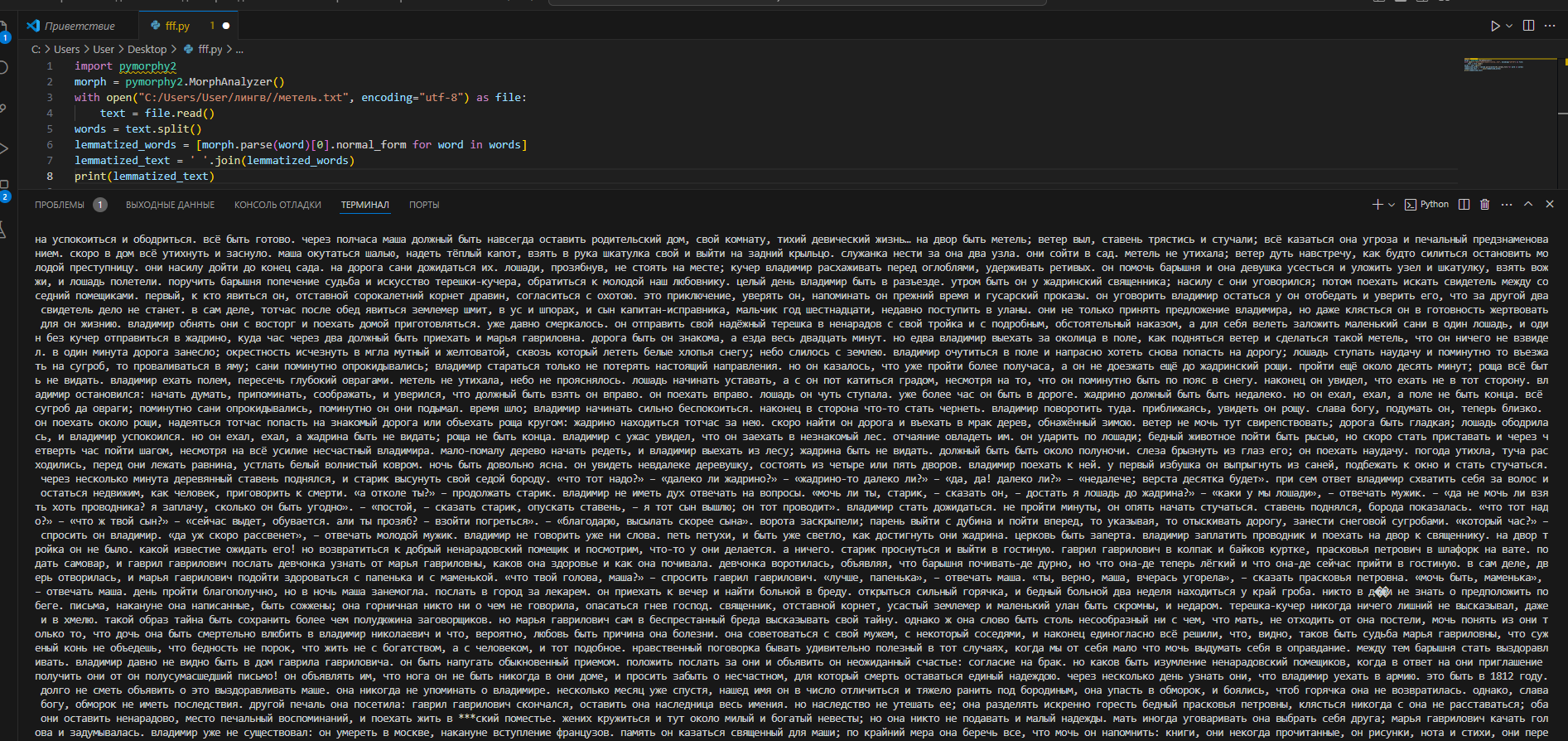
**Проведем стемминг  текста при помощи библиотеки NLTK**

****

Мы используем RussianStemmer из библиотеки NLTK для русского языка. Мы токенизируем исходный текст с помощью word\_tokenize, затем проводим стемминг каждого слова в тексте с помощью метода stem() русского стеммера. Наконец, объединяем стеммированные слова обратно в строку stemmed\_text.

После выполнения этого кода переменная stemmed\_text будет содержать текст, в котором каждое слово приведено к его основе (стему) согласно правилам русского языка.

**Проведем лемматизацию при помощи pymorphy2**

****

Мы используем MorphAnalyzer из библиотеки PyMorphy2 для проведения лемматизации слов. Мы разбиваем исходный текст на слова, затем для каждого слова проводим морфологический анализ с помощью morph.parse(word), выбираем первый вариант разбора [0] и получаем нормальную форму слова с помощью .normal\_form.

После выполнения этого кода переменная lemmatized\_text будет содержать текст, в котором каждое слово приведено к его лемме (нормальной форме).

Стемминг и лемматизация — это два различных метода нормализации текста в обработке естественного языка, но они имеют различия в том, как они обрабатывают слова.

1. Стемминг:

- Цель: Усечение слов до их основы (стема) путем удаления аффиксов.

- Пример: Для слов "пишу", "писал", "писать" результат стемминга будет одинаковым — "пис".

- Применение: Стеммеры работают по жестким правилам и обычно быстрее лемматизации, но могут приводить к потере смысла из-за усечения слов до корня.

2. Лемматизация:

- Цель: Приведение слов к их нормальной (словарной) форме.

- Пример: Для слов "пишу", "писал", "писать" результат лемматизации будет разным: "писать".

- Применение: Лемматизация использует словарь и грамматический анализ для приведения слов к их базовой форме, что помогает сохранить смысл текста.

Отличия:

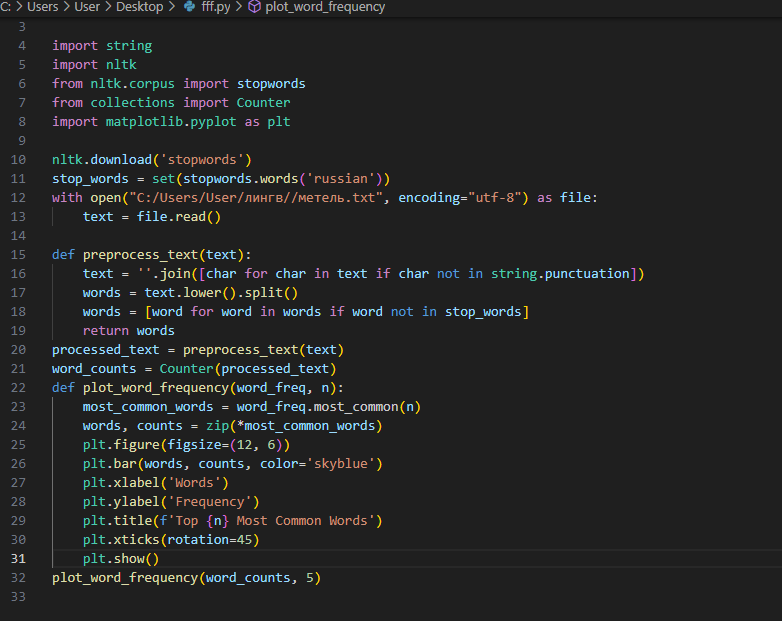
- Стемминг может привести к несуществующим или некорректным словам, так как он просто обрезает аффиксы, в то время как лемматизация приводит к реальным словам из словаря.

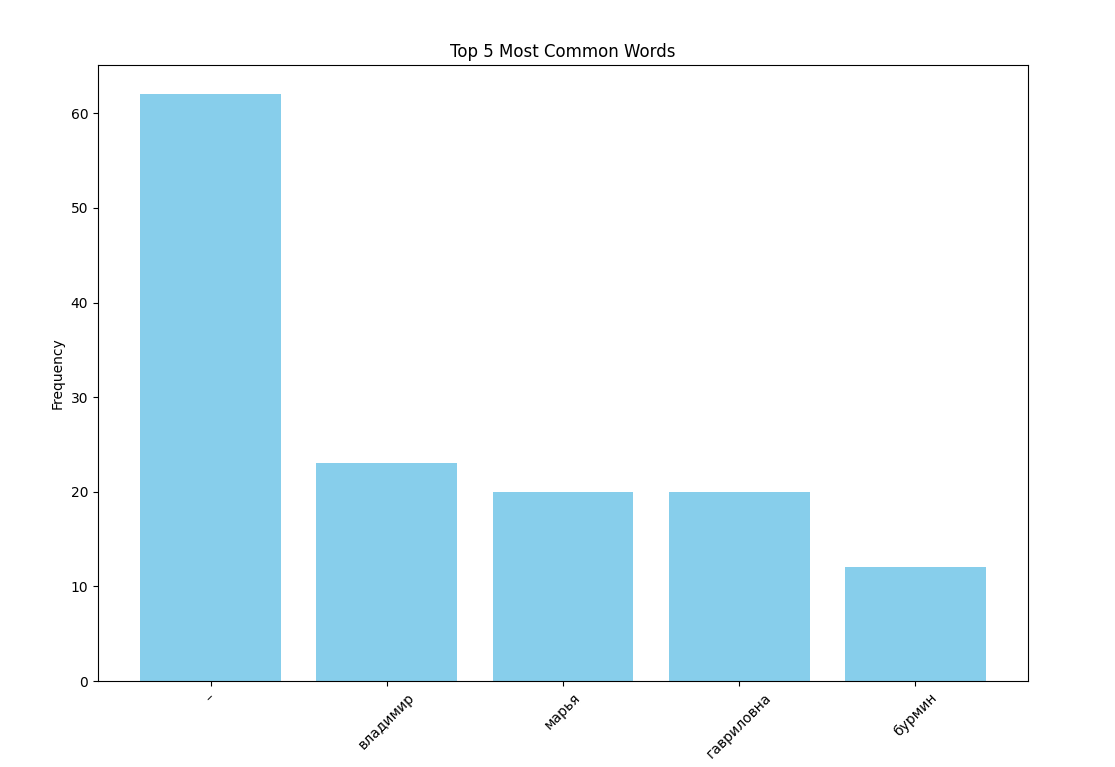
- Лемматизация более сложный и точный процесс, чем стемминг, так как она учитывает грамматические правила языка.

- Лемматизация обычно требует больше времени и ресурсов, чем стемминг, из-за более сложного алгоритма.

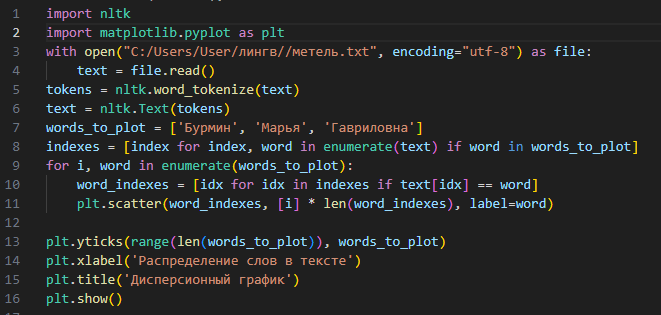
В общем, если важно сохранить смысл текста и работать с реальными словами из словаря, то лемматизация предпочтительнее стемминга.

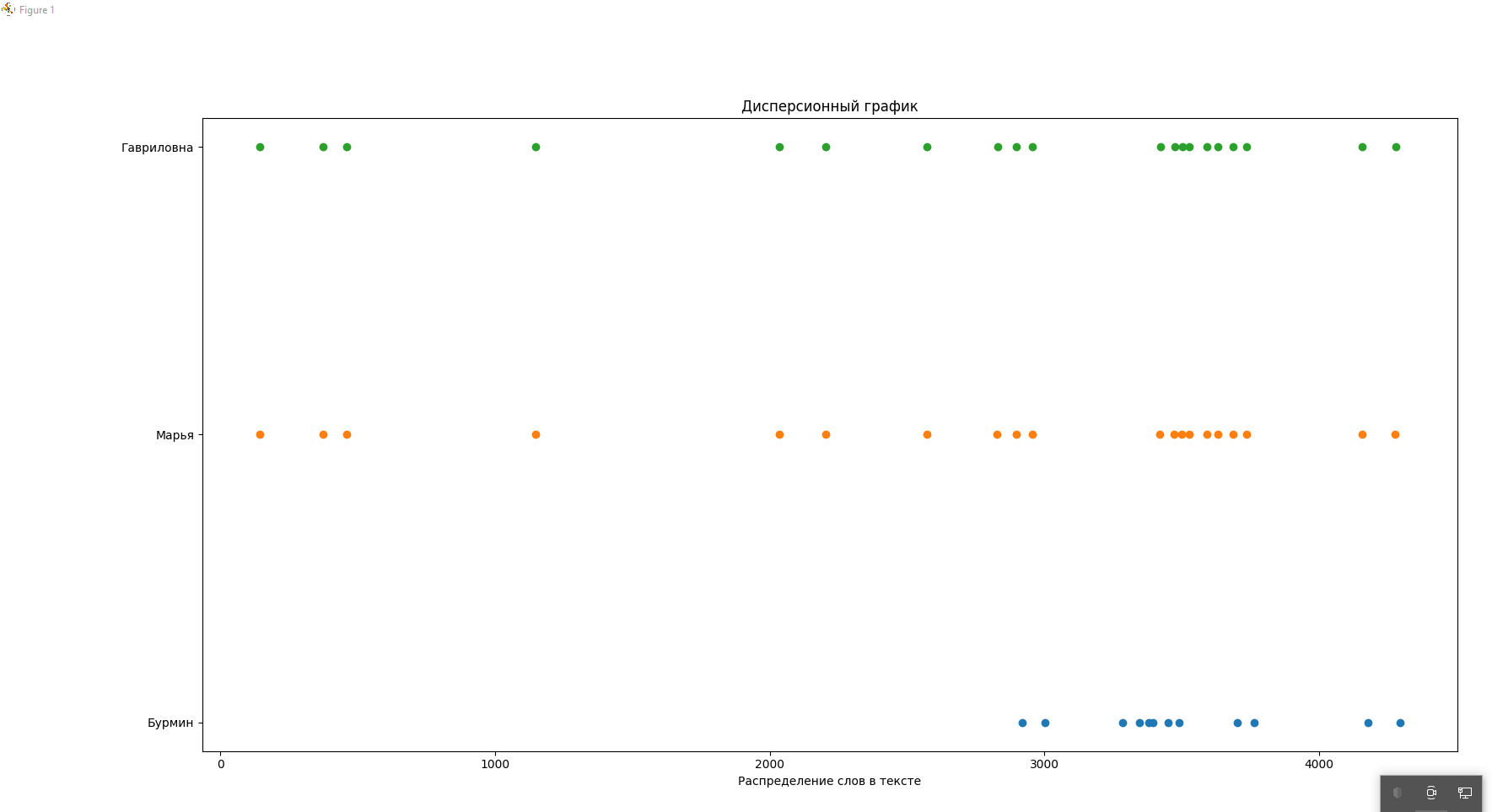
**Построем график наиболее частотных (лемматизированных) слов текста при помощи matplotlib**



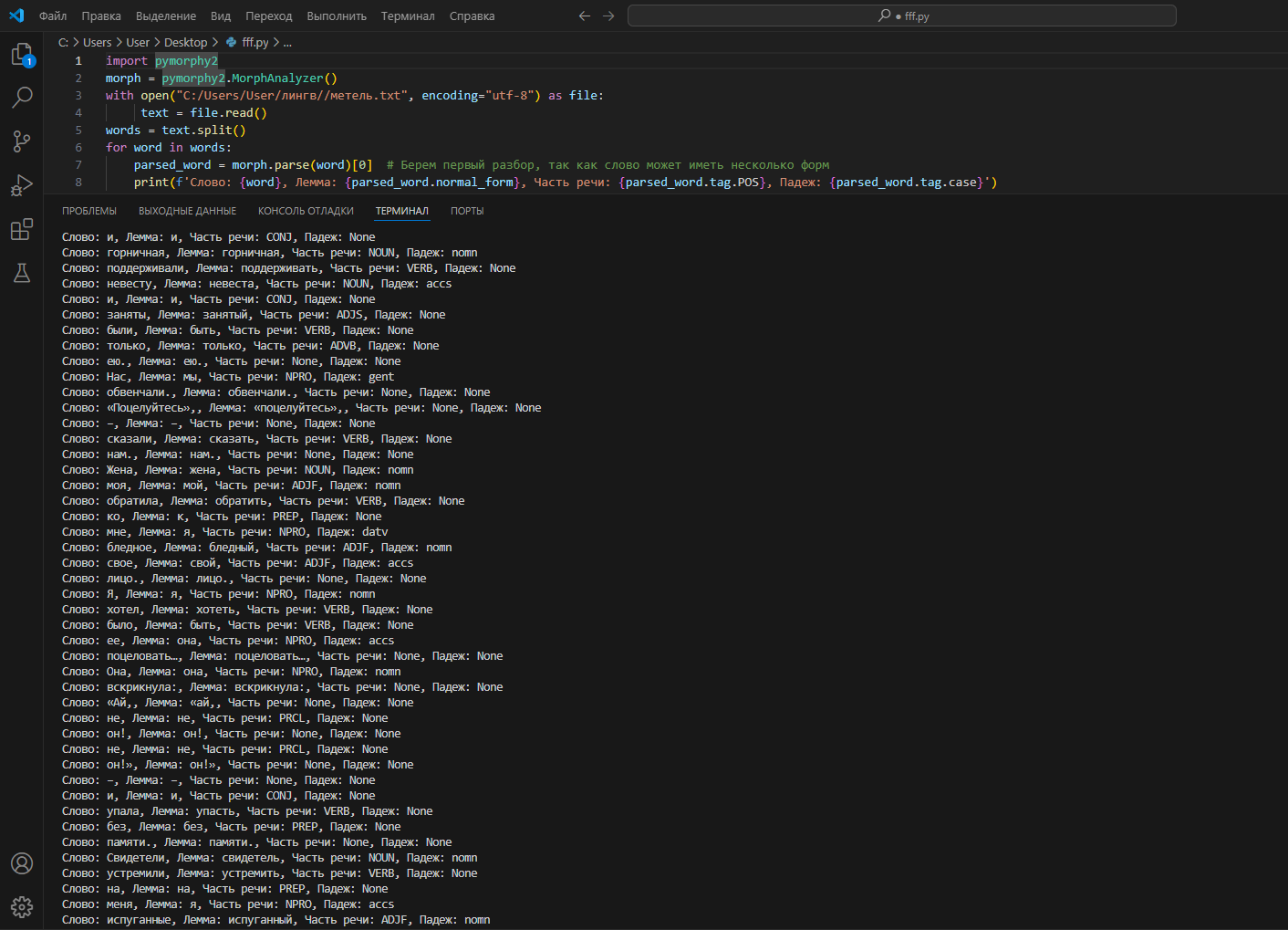


**Построем dispersion plot при помощи одноименного метода из библиотеки NLTK**

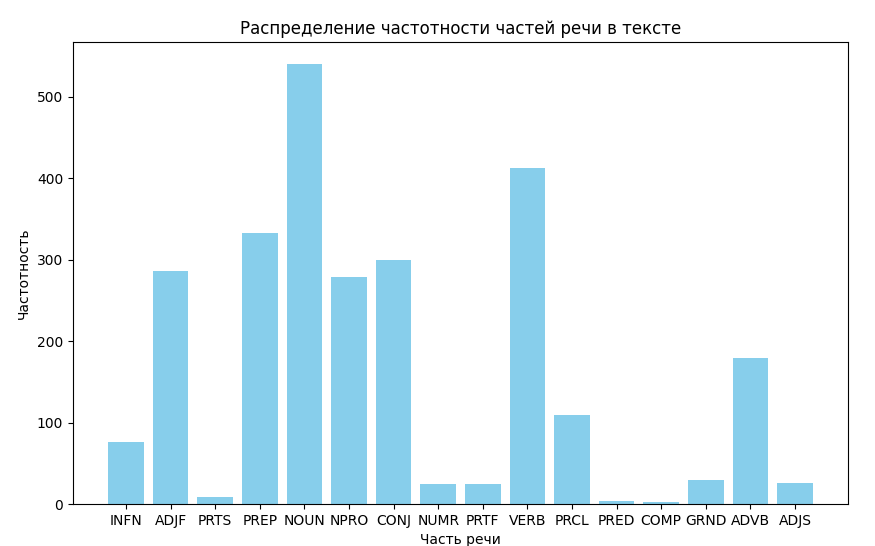


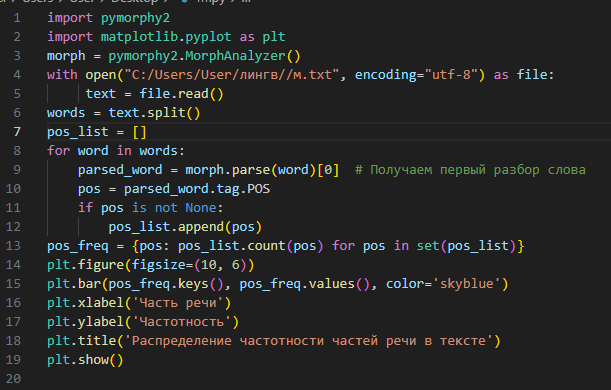


**Определим грамматические характеристики слов при помощи pymorphy2**

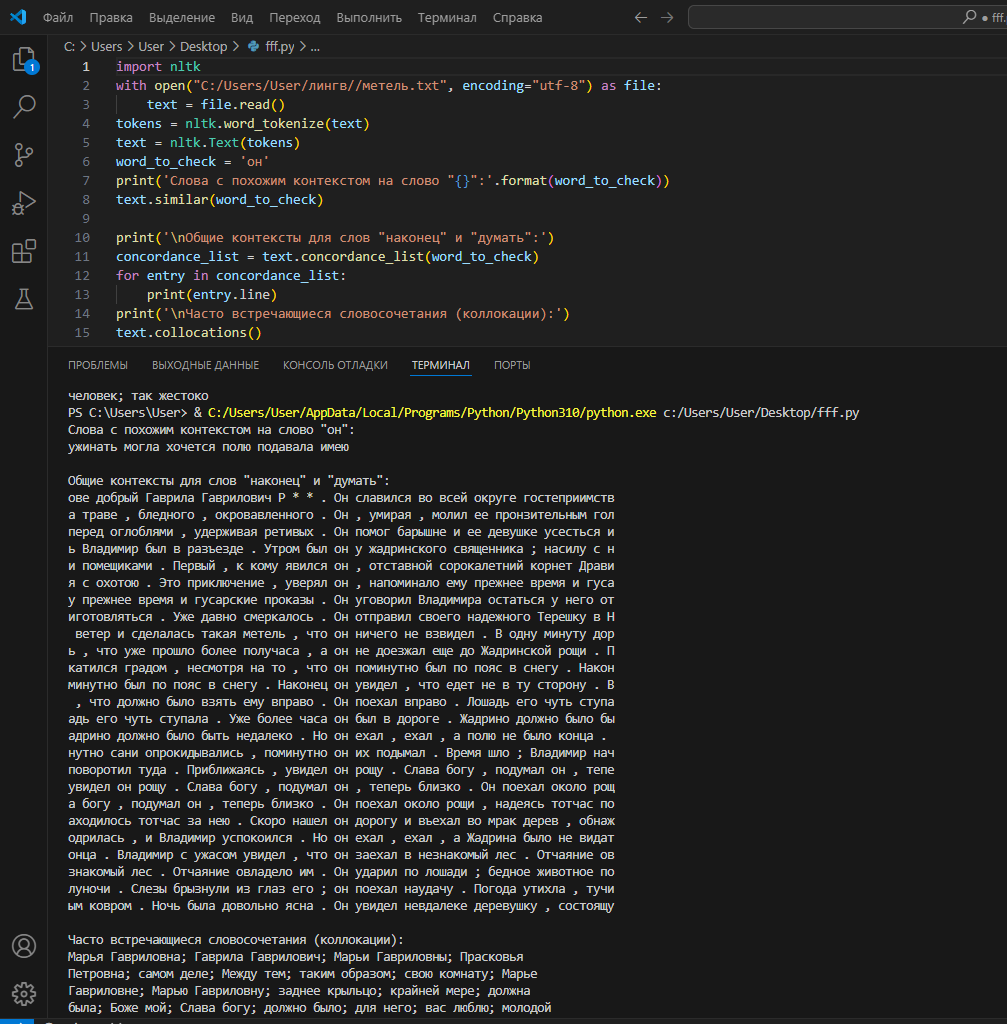
****

**Построим график, отражающий распределение частотности частей речи в тексте, при помощи matplotlib**

****

****

**Протестируем методы similar, common\_contexts и collocations из библиотеки NLTK**

****

Методы similar(), common\_contexts() и collocations() из библиотеки NLTK предоставляют различные возможности для анализа текста.

1. Метод similar(): Этот метод позволяет найти слова, которые встречаются в похожих контекстах в тексте. Например, если вы используете метод similar('слово'), он вернет список слов, которые встречаются в сходных контекстах с указанным словом.

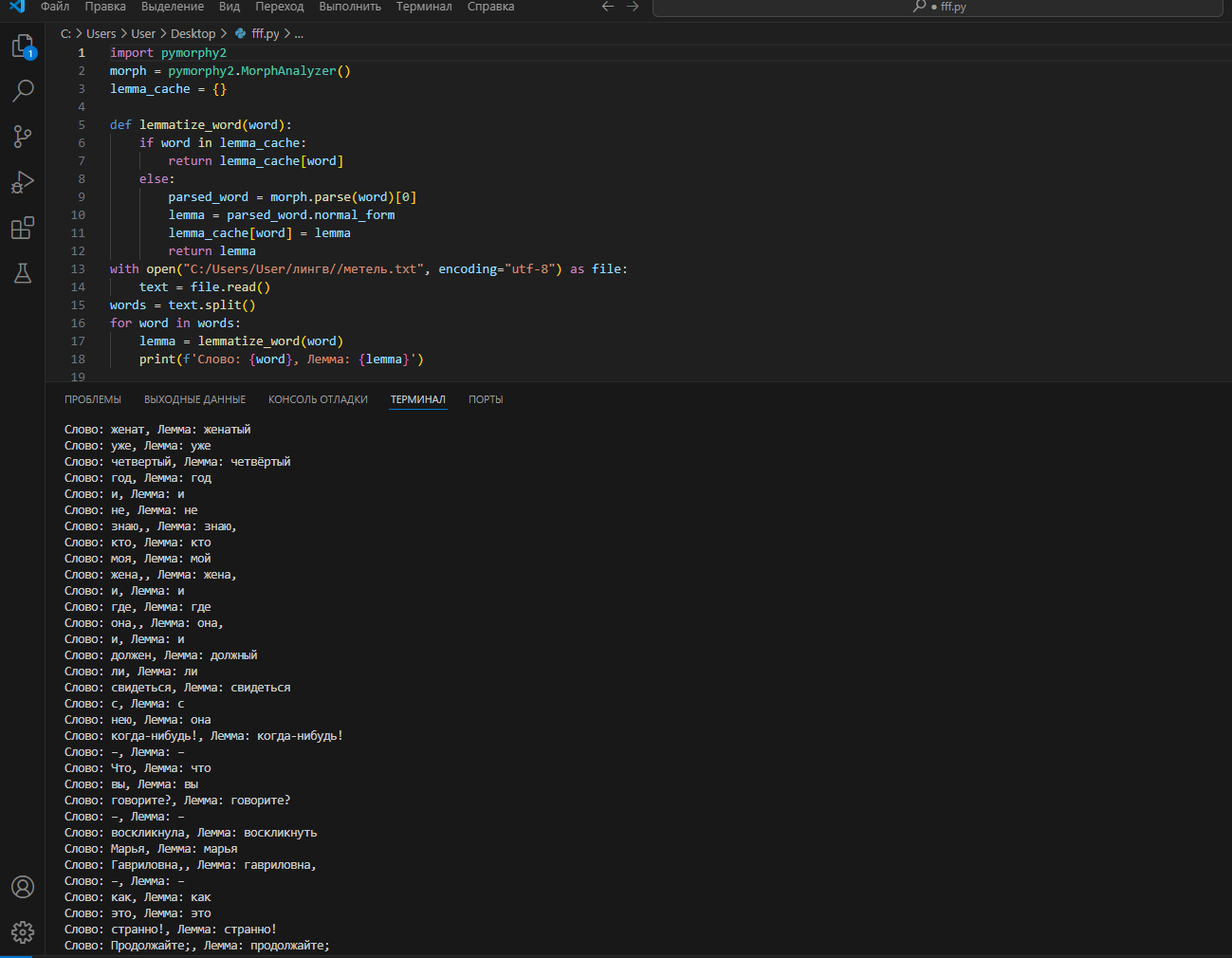
2. Метод common\_contexts(): Этот метод позволяет найти общие контексты для двух или более слов. Он выводит контексты, в которых указанные слова встречаются вместе. Например, common\_contexts(['слово1', 'слово2']) вернет контексты, в которых оба слова появляются.

3. Метод collocations(): Этот метод позволяет найти коллокации в тексте, то есть пары слов, которые часто встречаются вместе. Например, он может найти такие пары, как "быстро бежать" или "горячий чай".

Эти методы полезны для анализа текста, извлечения информации о контексте слов и выявления структурных особенностей языка. Они могут быть использованы для лингвистического анализа, изучения стилей письма, автоматического извлечения информации и других задач обработки естественного языка.

**Уровень 2**

**Проведите кэширование лемм.**

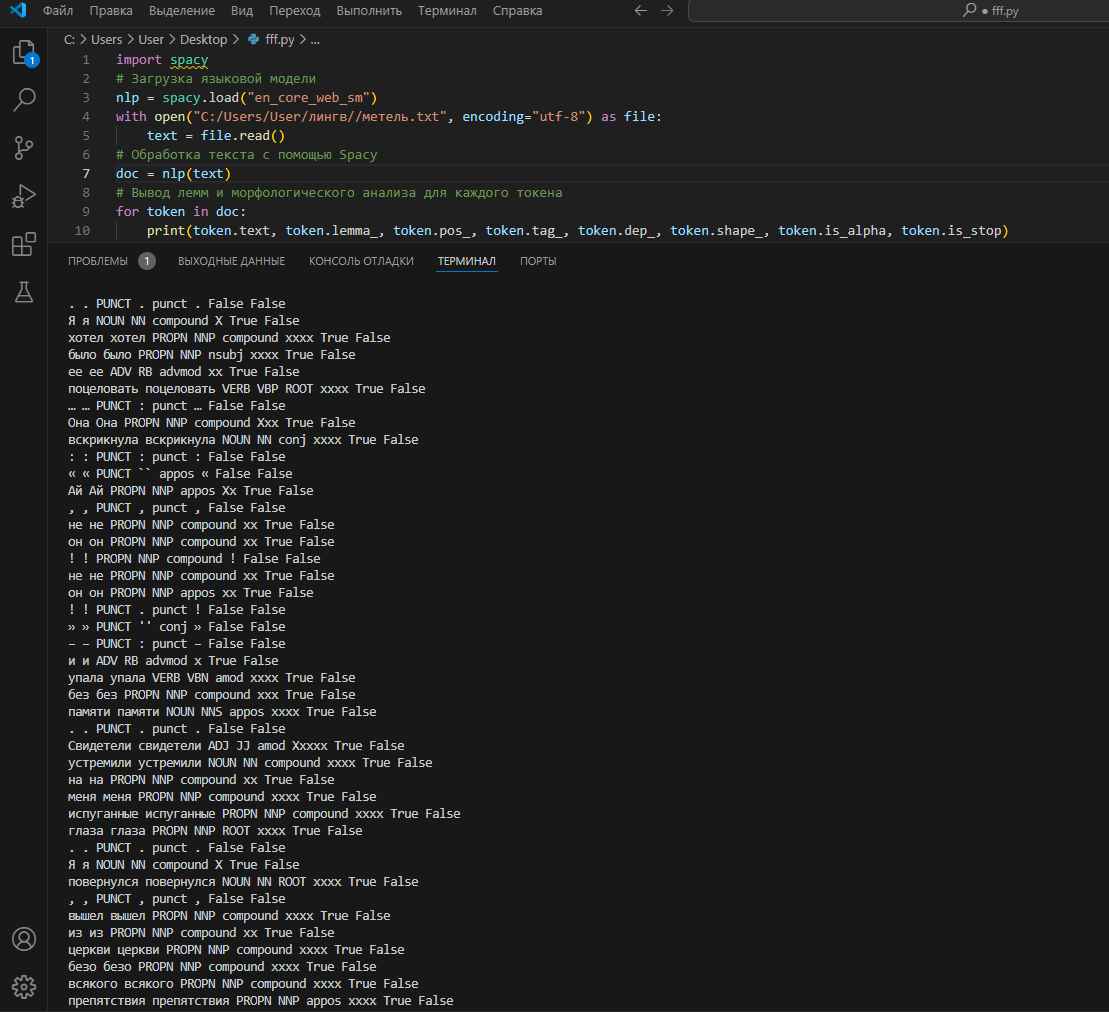
****

Кэширование лемм - это процесс сохранения результатов лемматизации слов или текста для последующего быстрого доступа к ним. Это может быть полезно в различных приложениях обработки естественного языка, таких как поисковые системы, машинный перевод, анализ тональности и другие.

Основная цель кэширования лемм состоит в том, чтобы уменьшить время выполнения операций лемматизации на больших объемах текста. При повторном обращении к уже лемматизированным словам или тексту кэш позволяет избежать повторных вычислений и сразу вернуть ранее полученный результат. Это особенно полезно в случаях, когда одни и те же слова или фразы встречаются неоднократно в тексте или запросах.

Другими словами, кэширование лемм помогает оптимизировать процесс обработки текста, сокращая время выполнения и ресурсы, необходимые для лемматизации. Таким образом, оно повышает эффективность работы приложений обработки естественного языка и общую производительность системы.

**Проведем лемматизацию и морфоанализ при помощи Spacy.**

****

Отличия между библиотеками Spacy и pymorphy2 включают в себя следующее:

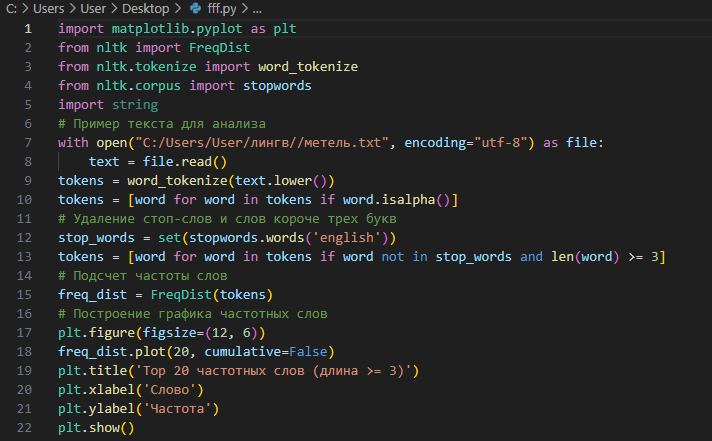
1. Обработка текста в целом и контекстуализированный анализ: Spacy работает со всем текстом как целым, стремясь к контекстуализированному анализу, в то время как pymorphy2 работает с каждым словом независимо от контекста.

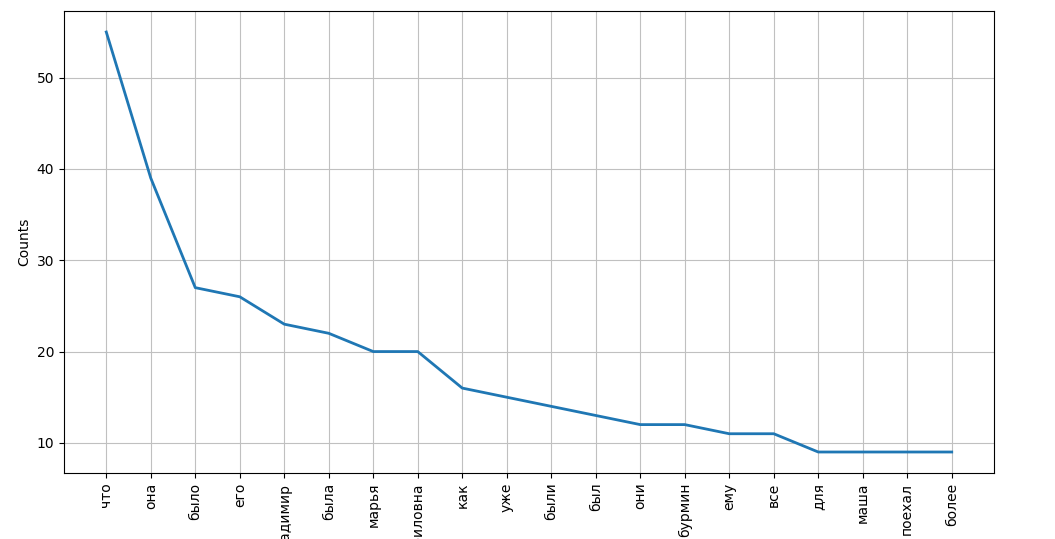
2. Множество языков и поддержка: Spacy поддерживает различные языки и имеет модели для более широкого спектра задач обработки текста, в то время как pymorphy2 сосредоточен на морфологическом анализе слов русского языка.

3. Интерфейс и удобство использования: Spacy предоставляет более удобный и гибкий интерфейс для обработки текста, предоставляя информацию о токенах (словах) и их свойствах напрямую, в то время как pymorphy2 предоставляет отдельные методы для морфоанализа и лемматизации.

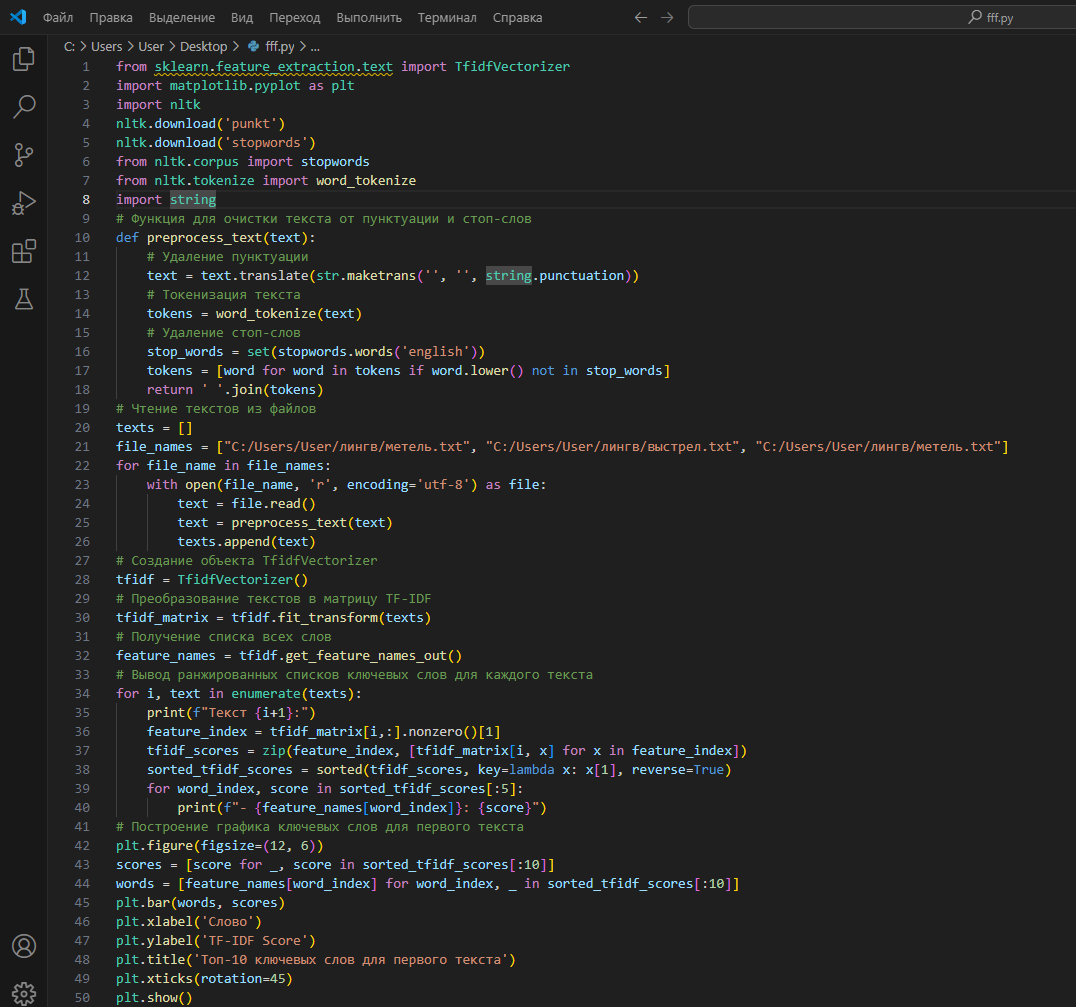
4. Скорость и производительность: Spacy обычно быстрее в работе с большими объемами текста, благодаря применению различных оптимизаций и использованию современных методов обработки текста.

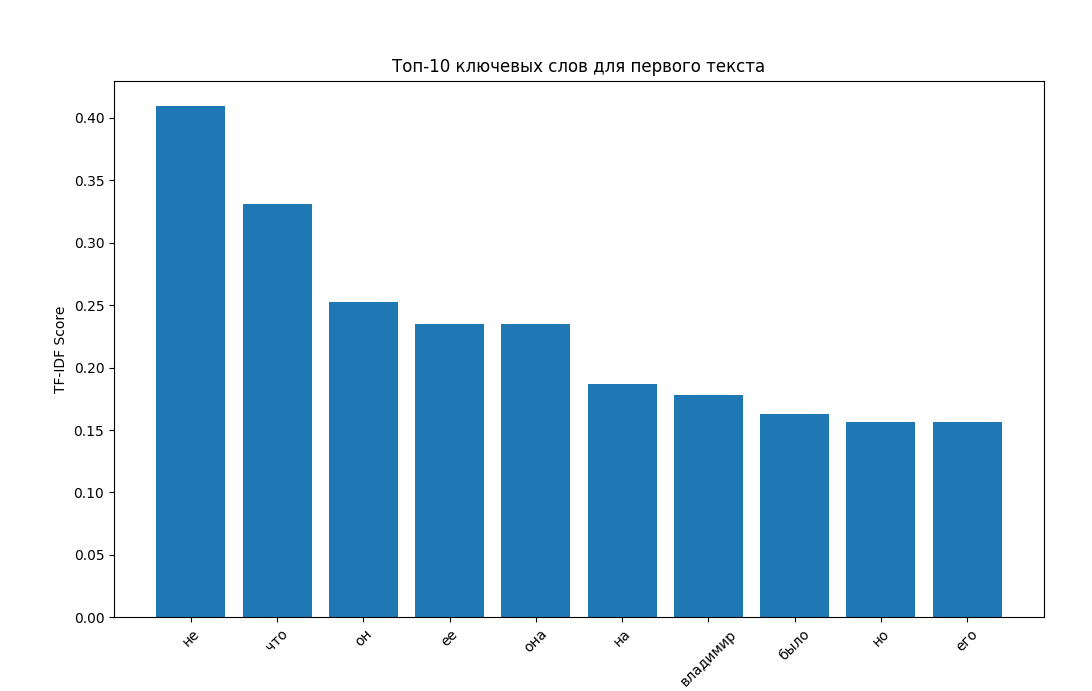
**Постройте график частотных слов, удалив из него все слова короче 3х букв.**

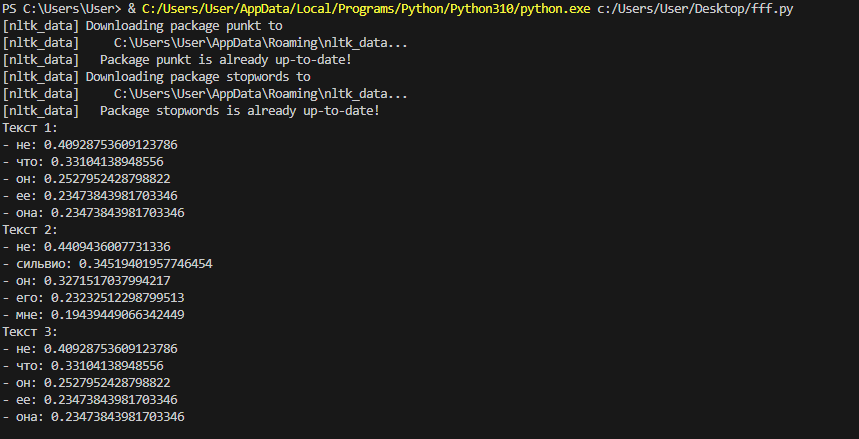
****

****

**Посчитайте tf-idf для 3х текстов (тексты не должны совпадать). Выведите ранжированные списки ключевых слов для каждого текста. Постройте график ключевых слов для одного из текстов.**

****

****

****

**Конспект лекции**

Произвольность языкового знака, что это означает что отличие от знаков от тех, которые стоят на дороге языковые знаки в том числе слова, предложение, тексты произвольные у них форма не связано напрямую с содержанием, не связано напрямую с теми образами теме концептами, которые стоят за ними именно поэтому одни и те же образы, одни и те же объекты в разных языках выражаются совершенно разными цепочками, звуков цепочками букв. В анализе языка вообще все построено на разбиение на уровне и вот для нижележащих уровней языка компьютерная логистика в принципе научилась производя производить моделирование, то и разбивать эти нижележащих уровней на какие-то, строить какие-то модели: фонетика, морфология, синтаксис.

Моделирование – это когда мы можем в сжатом виде представить некие формы проводимого исследования.

Нужно смоделировать, у каких слов схожее значение. То есть придумать такие репрезентации слов, что если слово одно и тоже, то и репрезентации должны быть похожими.

Фундаментальные подходы к построению семантики: построение онтологий, извлечения значений из употребляемых слов в тексте.

Дистрибуция – распределение явлений в живой речи.

Способ представления значений в дистрибутивной семантике – семантические вектора.

При создании матрицы совместной встречаемости можно смотреть не только на непосредственных соседей, но и на слова на некотором расстоянии. Размер контекста – 2, 3 слова.

Определение семантической близости слов в векторном пространстве – через косинусную близость векторов.

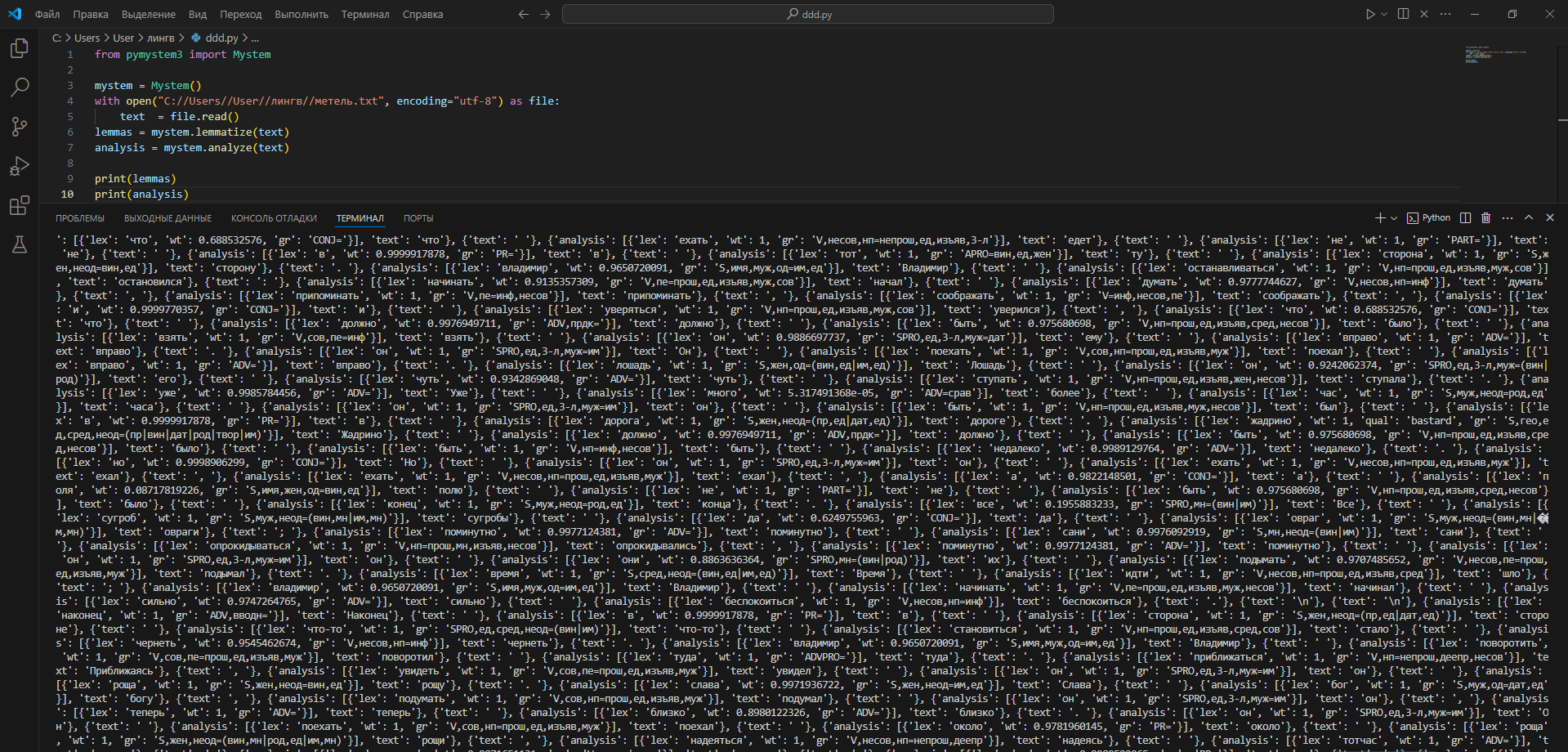
Недостатки счетных моделей: размеры векторов получаются огромными, что замедляет операцию сравнения векторов, страдает качество при использовании разных методов снижения размерности, нельзя определить, что является нужной информацией, а что мусором.

Популярный метод машинного обучения в языковых моделях – исскуственные нейронные сети.

В дистрибутивной семантике модели на основе машинного обучения называются предсказательными.

**Уровень 3**

**Проведите лемматизацию и морфоанализ при помощи mystem в Питоне. Каковы отличия от pymorphy2**

****

1. pymorphy2:

- Pymorphy2 является модулем для морфологического анализа русских слов. Он основан на словаре и правилах и предоставляет информацию о форме слова, его части речи, падеже, числе и других морфологических характеристиках.

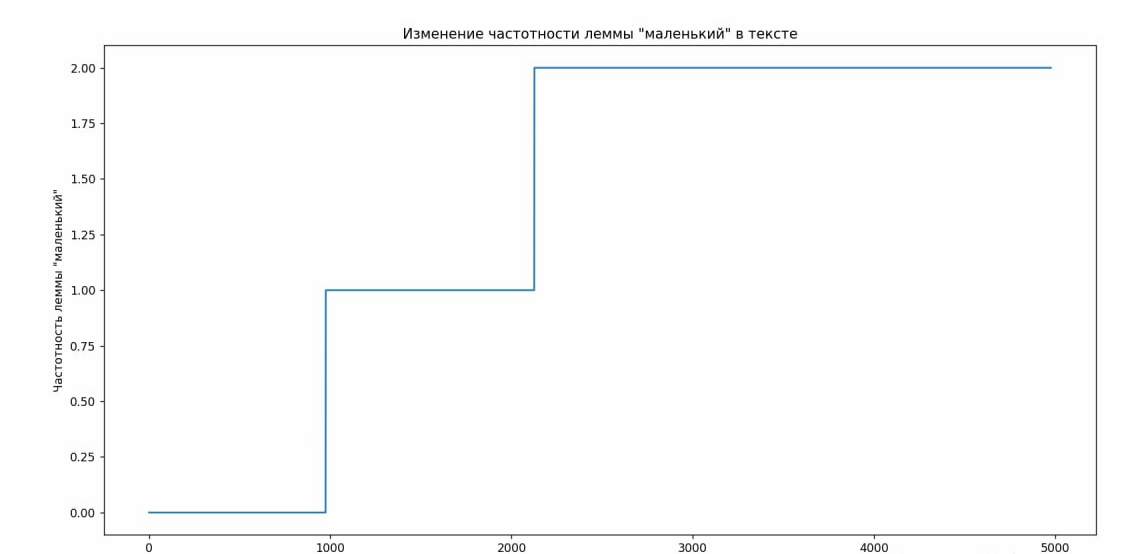
- Основное отличие от Mystem заключается в том, что Pymorphy2 работает на основе словаря и правил, в то время как Mystem использует статистические методы.

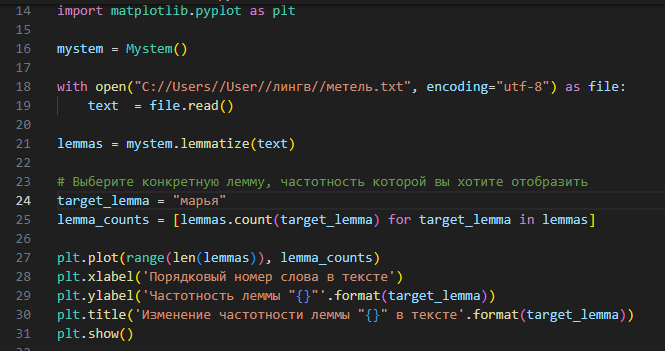
2. Spacy:

- Spacy - это библиотека для обработки естественного языка, предоставляющая возможности токенизации, лемматизации, извлечения сущностей и синтаксического анализа.

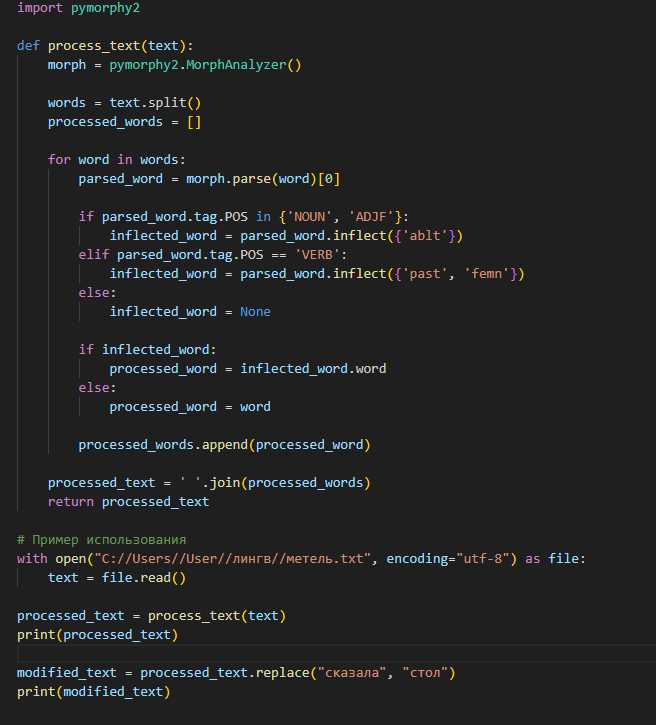
- Отличие от Mystem заключается в том, что Spacy предоставляет более широкий спектр инструментов для обработки текста, включая поддержку нескольких языков и возможность работы с различными компонентами обработки естественного языка.

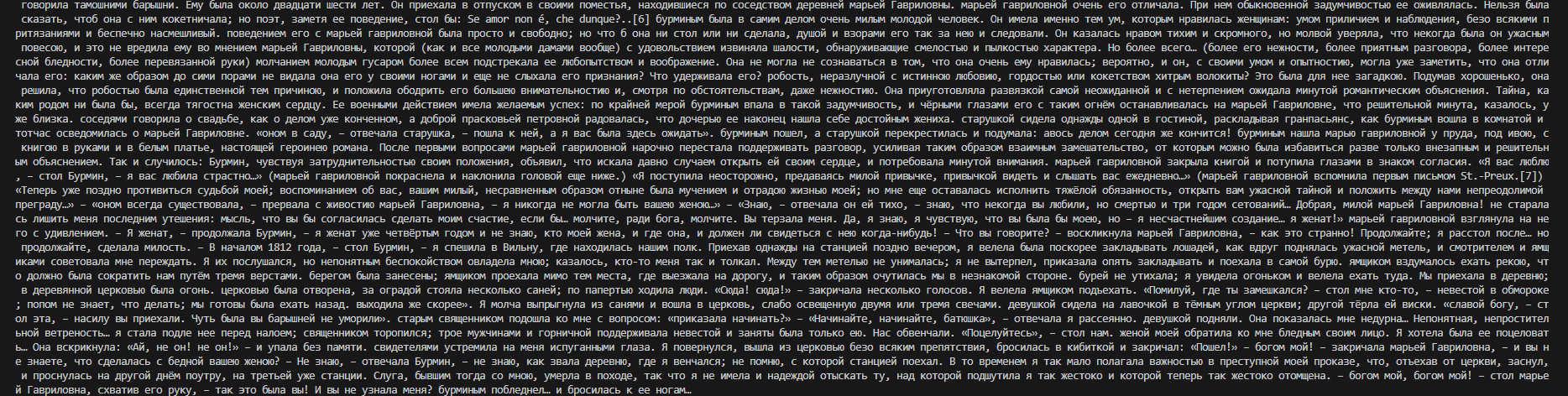
**Отразите на графике изменения частотности конкретной леммы по тексту (как в Google Ngram Viewer).**

****

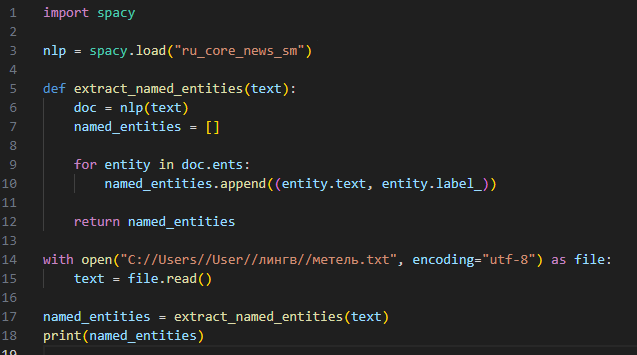
****

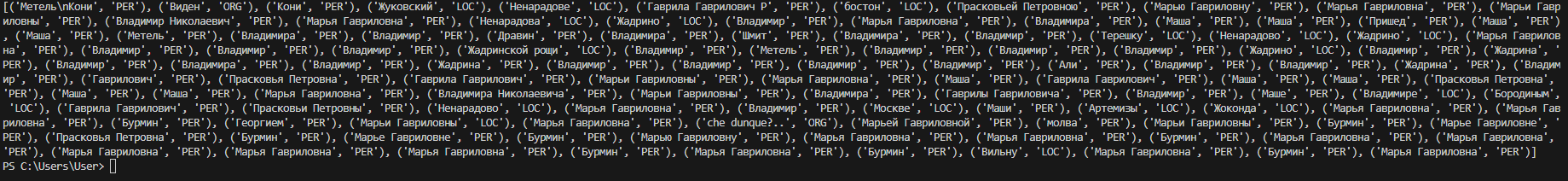
**Используйте pymorphy2 для морфологического синтеза:  
- Напишите код, который для любого текста ставит все существительные и прилагательные в творительный падеж, а все глаголы в прошедшее время и женский род  
- Замените ряд существительных в вашем тексте (например, всех лошадей на черепах); характеристики слов должны остаться грамматически верными.**

****

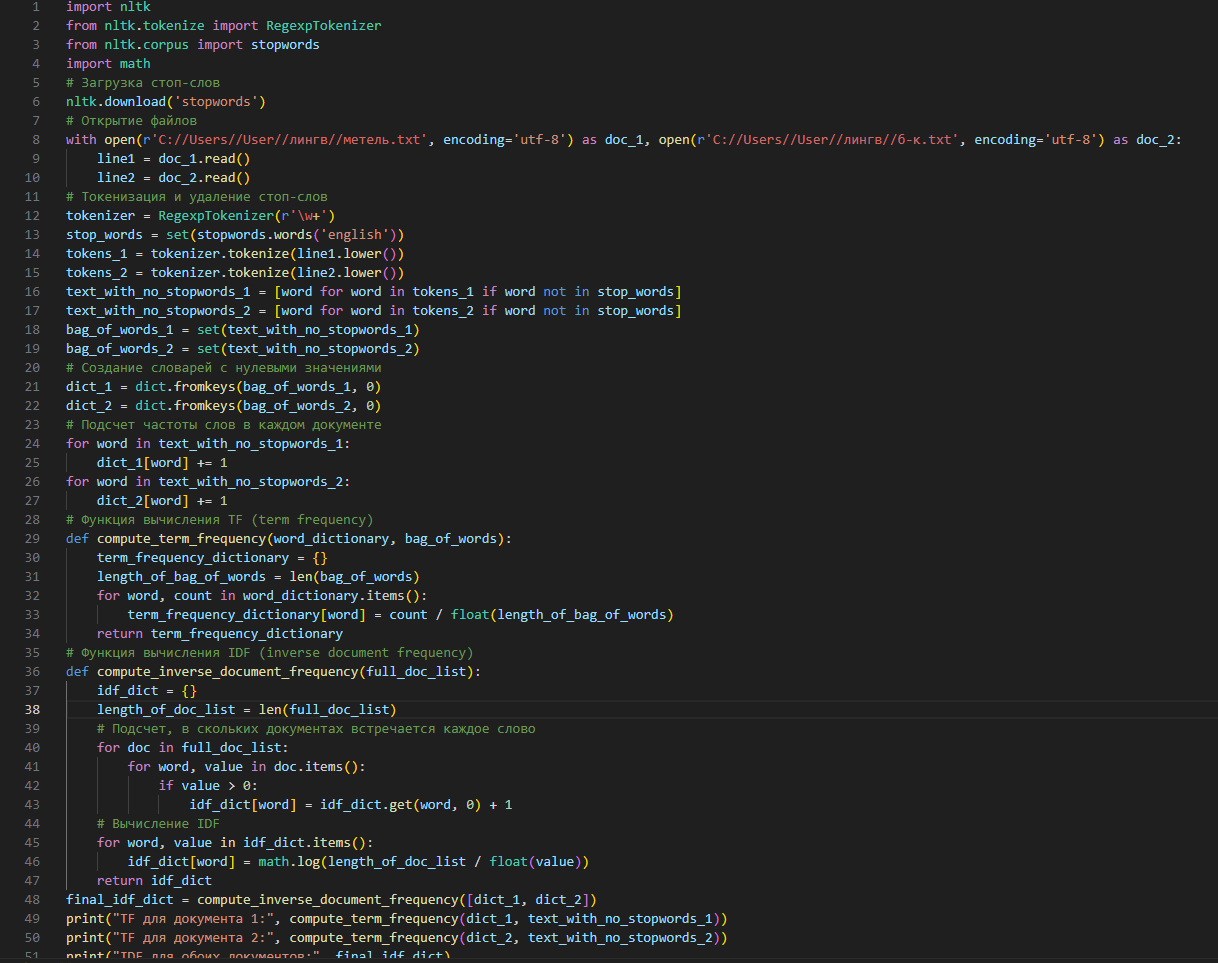
****

**Извлеките именованные сущности из вашего текста при помощи Spacy.**





**Исправьте ошибки в коде по пошаговому расчету tf-idf из файла "**[**Лабораторная tf-idf (с ошибками)**](https://edu.vsu.ru/mod/url/view.php?id=1102325)**" (не забудьте сохранить копию блокнота на своем диске перед внесением изменений)**

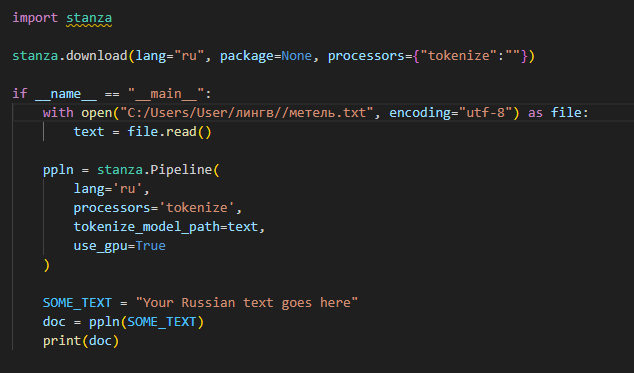
****

1. При открытии файлов в коде использовали разные переменные для хранения текста из файлов, но затем переписали переменную text, из-за чего потеряли доступ к содержимому файлов. Исправили это, чтобы текст из обоих файлов сохранялся в переменной raw.

2. В функции compute\_term\_frequency передали неправильные аргументы при вызове функции print.

3. В функции compute\_inverse\_document\_frequency исправили ошибку в расчете значения IDF для каждого слова.****

**Протестируйте нейросетевые модели НКРЯ: https://ruscorpora.ru/license-content/neuromodels/ Проанализируйте и сравните результаты.**

****

****