**Задание В.**

Токенизация (tokenization) — это процесс разбиения текста на отдельные слова или другие единицы (токены). В естественном языке, таком как русский, слова обычно разделяются пробелами или знаками пунктуации, такими как запятые или точки. Однако в некоторых случаях может быть не очевидно, где заканчивается одно слово и начинается другое, например, когда слово состоит из нескольких частей или когда оно содержит знаки препинания, например, такое может встречаться в сокращениях («т.д.», «т.п.»).

Токенизация может быть необходима во многих задачах обработки естественного языка, таких как анализ тональности, машинный перевод, распознавание именованных сущностей, и многих других. Как правило, перед обработкой текста, он должен быть токенизирован, чтобы алгоритмы могли корректно обрабатывать каждое слово или другую единицу текста в отдельности. После токенизации часто проводится лемматизация (приведение слов к нормальной форме). В рамках данного задания необходимо реализовать лишь токенизацию. Для данной задачи было принято использовать язык программирования Python. Код реализован в редакторе «jupyter notebook», файл с решением прикреплен к заданию.

Код выполняет разбиение на предложения и токенизацию каждого преждожения с помощью регулярных выражений. Код протестирован на реальных текстах из разных предметных областей и жанров. Для тесирования выбраны отрывки из научной статьи, новостной статьи и художественного произведения. Кроме того, в рамках данной работы написанный на основе правил токенизатор сравнивается с готовым токенизатором из сторонней библиотеки для обработки ествественного языка — «nltk».

Существуют различные методы токенизации, в том числе методы, основанные на правилах (например, с использованием регулярных выражений), а также методы, основанные на машинном обучении, которые используются для автоматического извлечения правил на основе большого объема обучающих данных.

**Реализация собственного токенизатора**

Для создания графематического анализатора на основе правил, использоваkbcm регулярные выражения (библиотека re). Например, можно использовать следующий код на Python для токенизации текста на основе простых правил:

import re

text = "Это пример текста для токенизации. Он состоит из нескольких предложений."

tokens = re.findall(r'\w+', text)

print(tokens)

В данном примере используется регулярное выражение «\w+» для поиска слов в тексте. Это выражение находит все подстроки, состоящие из букв, цифр и символа подчеркивания. Функция re.findall() возвращает список всех совпадений с регулярным выражением.

Однако такой подход может дать не совсем точные результаты, так как русский язык имеет свои особенности, в тексте часто встречаются сокращения, дефисы и т.д., поэтому мы можем использовать более сложные правила для токенизации текста на русском языке. Например, может подойти следующее регулярное выражение:

\b(?:[А-Яа-яЁё]\.)+[А-Яа-яЁё]?\.?|\b\w+(?:-\w+)\*\b

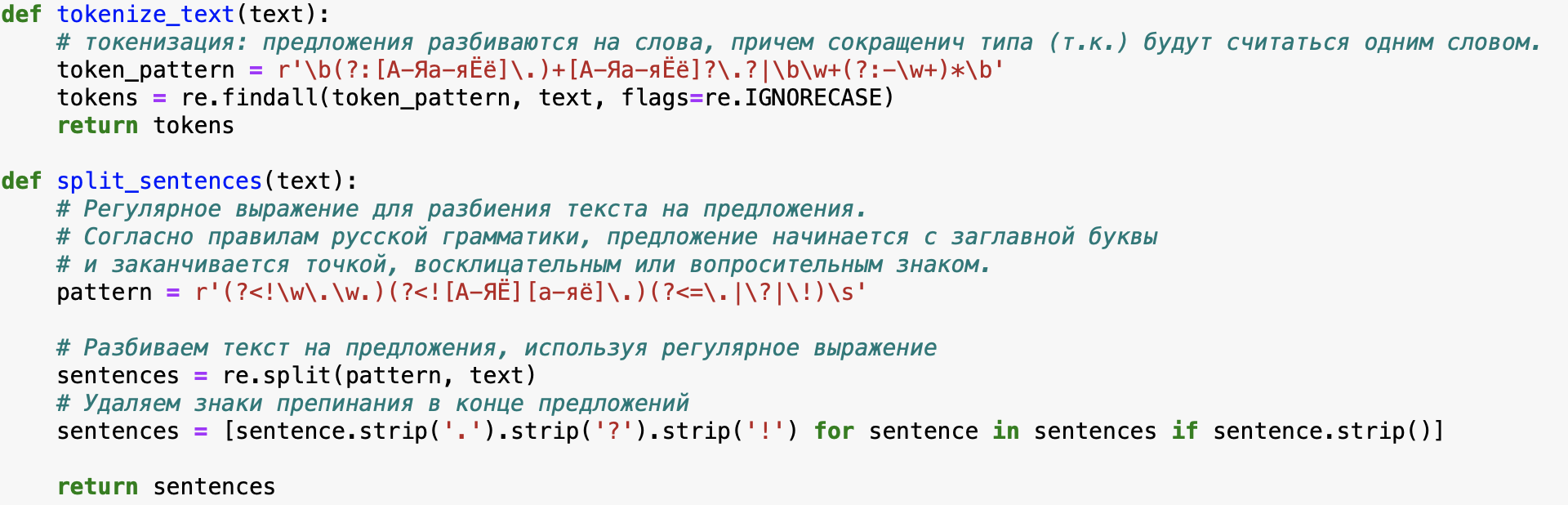


Рисунок 1. Пример реализации токенизации текста.

С помощью этого кода предложения можно разбить на слова, а сокращения типа (т.к.) будут считаться одним словом.

Регулярное выражение находит все подстроки, которые начинаются с границы слова (\b), состоят из букв, цифр и символа подчеркивания (\w+), и могут содержать дефисы, и заканчиваются границей слова (\b). Флаг «re.IGNORECASE» указывает, что поиск должен быть регистронезависимым.

Таким образом, мы можем использовать регулярные выражения для токенизации текста на русском языке, применяя соответствующие правила. Однако, для более точного анализа текста, особенно при работе с различными формами слов, может потребоваться применение более сложных методов обработки естественного языка.

Таким образом, текст разбивается на предложения, учитывая особенности русского языка:

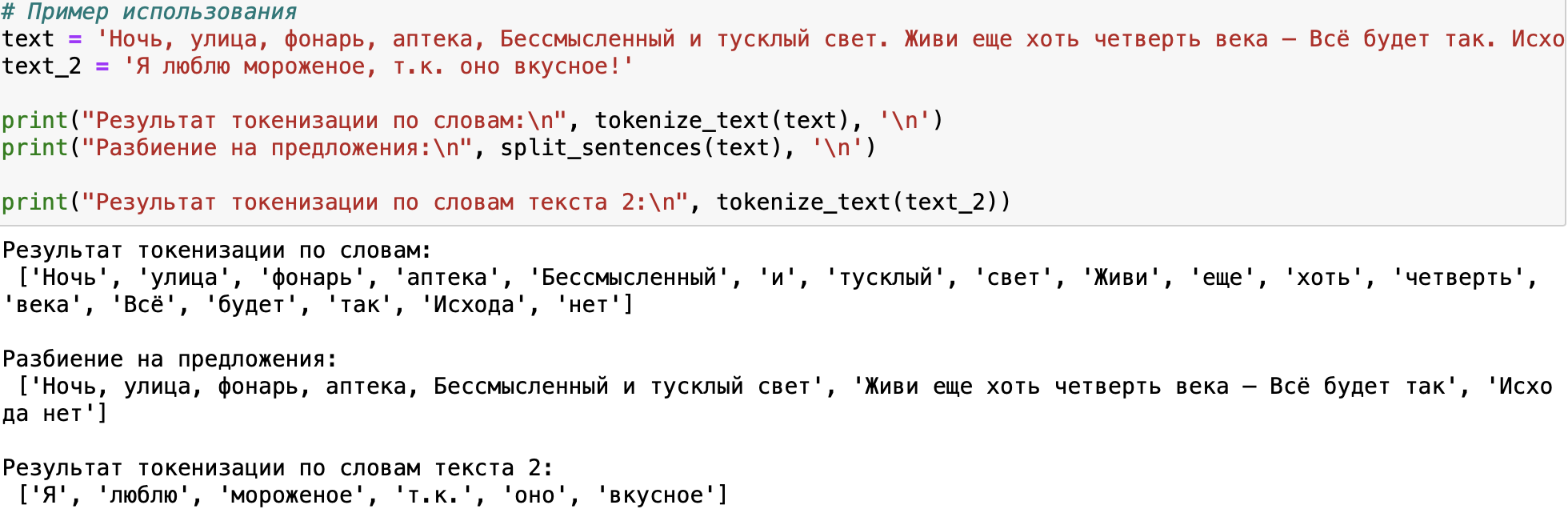


Рисунок 2. Пример токенизации текста с помощью созданной вручную функции

return tokens

**Реализация готового токенизатора из библиотеки nltk**

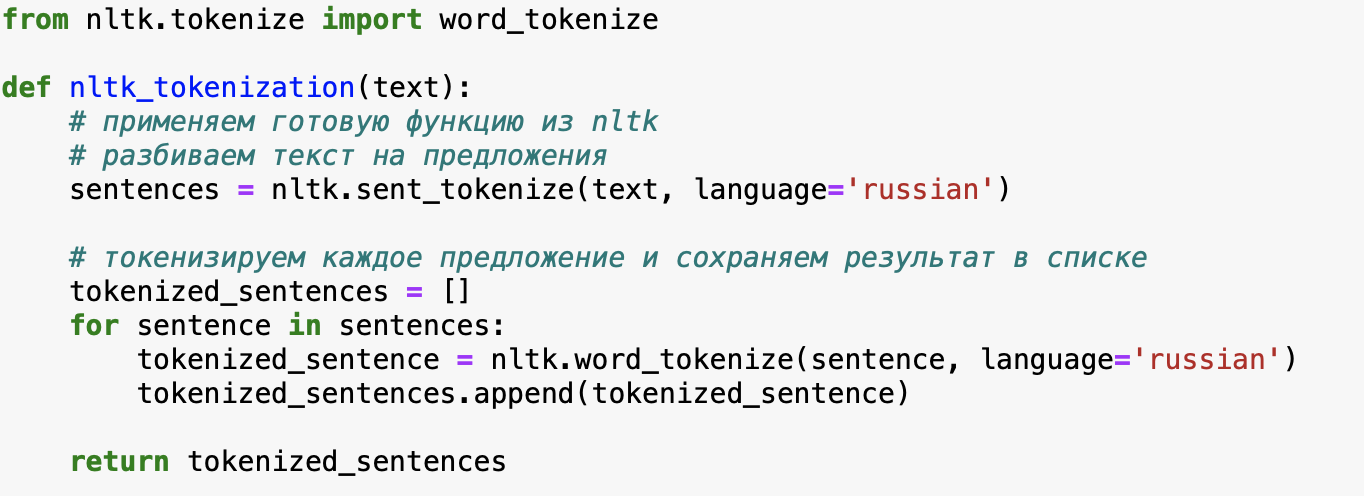
****

Рисунок 3. Использование готового решения

**Тестирование**

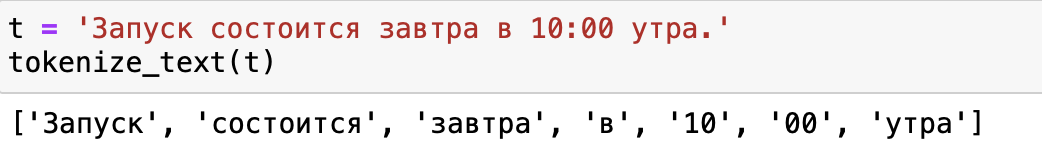
Для оценки точности токенизации и разбиения на предложения можно использовать метрику precision (долю верно определенных токенов/предложений). Для этого необходимо иметь эталонную токенизацию текстов, а затем сравнивать этот образец с результатом работы функций токенизации. В данном случае, мы можем оценить точность токенизации и разбиения на предложения и проверить, что функции работают корректно и находят нужные токены и предложения на реальных текстах трех стилей: научного, художественного и публицистического.

Для примера, возьмем следующий текст:

text = 'Запуск состоится завтра в 10:00 утра.'

Применим функцию «tokenize\_text»:

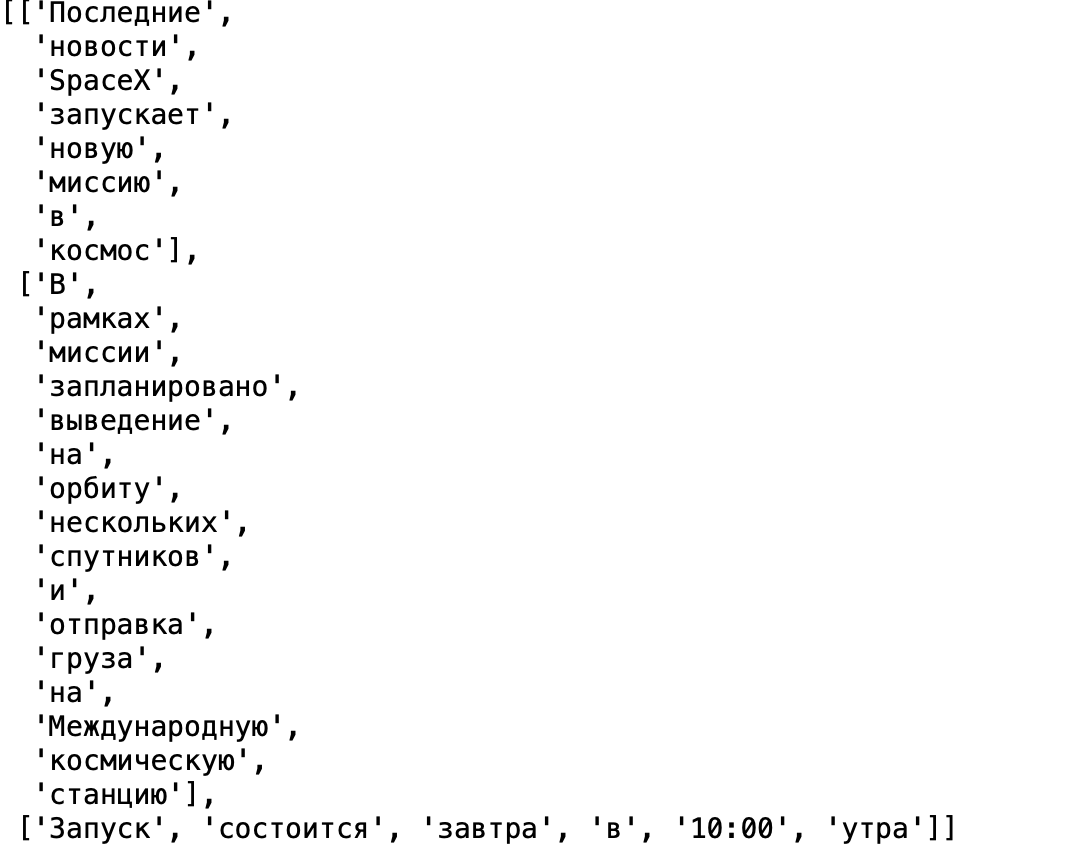
Результат:



Можно увидеть, что в данном случае наша функция «tokenize\_text» справляется не совсем успешно, разделяя время на два токена. Следует обратить на это внимание и отредактировать регулярное выражение внутри функции.

Выше мы уже проверяли, что функция хорошо справляется с сокращениями.

Теперь проверим, как справляются функции из библиотеки «nltk». Возьмем небольшой отрывок из научной статьи, разделим его на предложения, а затем на токены.



Похоже, что функции справились с задачей и нашли все нужные предложения в текстах и токены в них.

Чтобы лучше оценить точность работы функций, можно использовать какой-то набор заранее известных текстов большего объема.

Тексты будут приложены к работе в файлах формата «.txt». Кроме того, к работе прилагаются файлы, где каждый из текстов вручную разбит на предложения и слова. Таким образом, мы сможем качественно оценить работу нашего токенизатора, сравнив результат его работы с эталонной токенизацией.

Можно также использовать готовые корпуса текстов, которые уже разбиты на предложения и токены, чтобы сравнить результат работы своих функций с этими корпусами. Например, в русском языке такими корпусами могут быть корпуса НКРЯ (Национальный корпус русского языка) или Taiga Corpus.

Если результаты будут достаточно близки к результатам из этих корпусов, то можно считать, что функции работают достаточно точно. Если же будут ошибки или пропуски, то нужно будет дополнительно работать над улучшением алгоритмов.

Для оценки точности работы функций будем использовать заранее известные тексты и сравнивать результаты разбиения на предложения и токены, полученные с помощью функций, с эталонным разбиением.

Будем использовать следующие тексты:

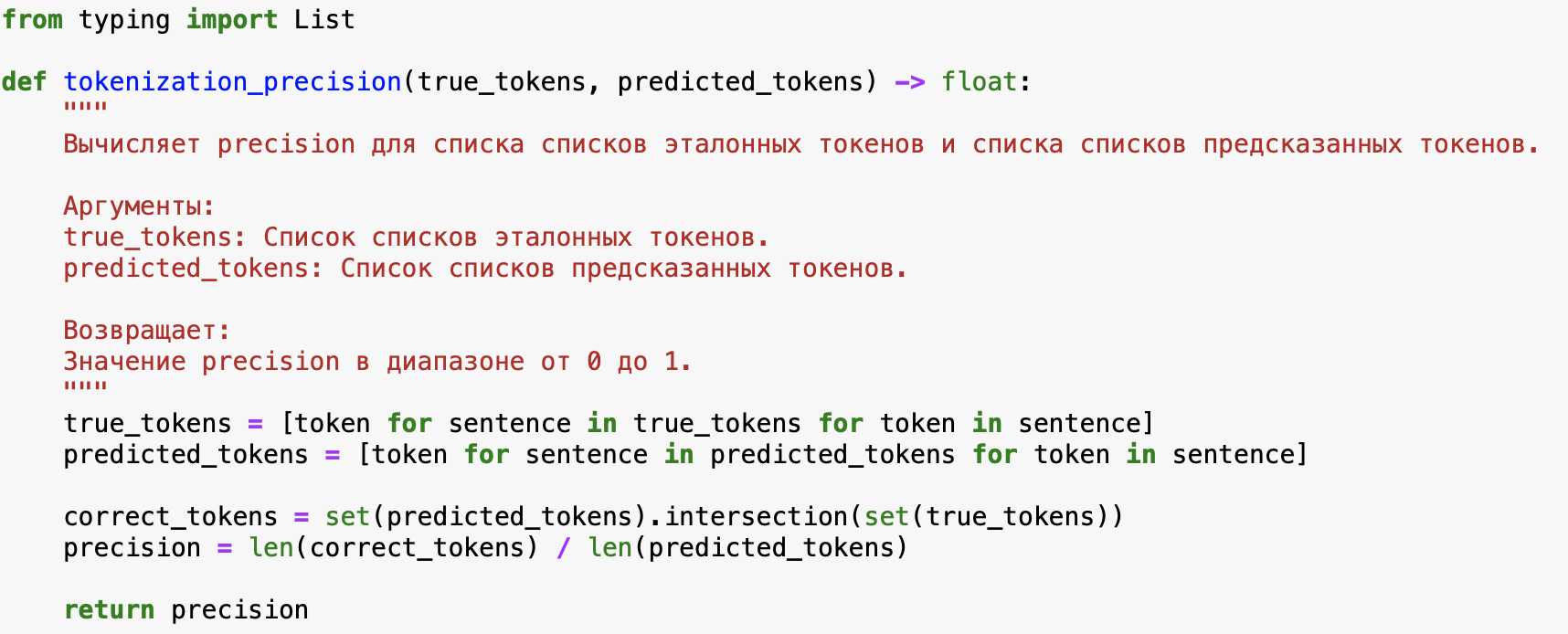
- Научная статья: "Бедность в России. Можно ли остановить этот процесс";

- Новостная статья: "Великобритания: путешествие по самым удивительным местам страны";

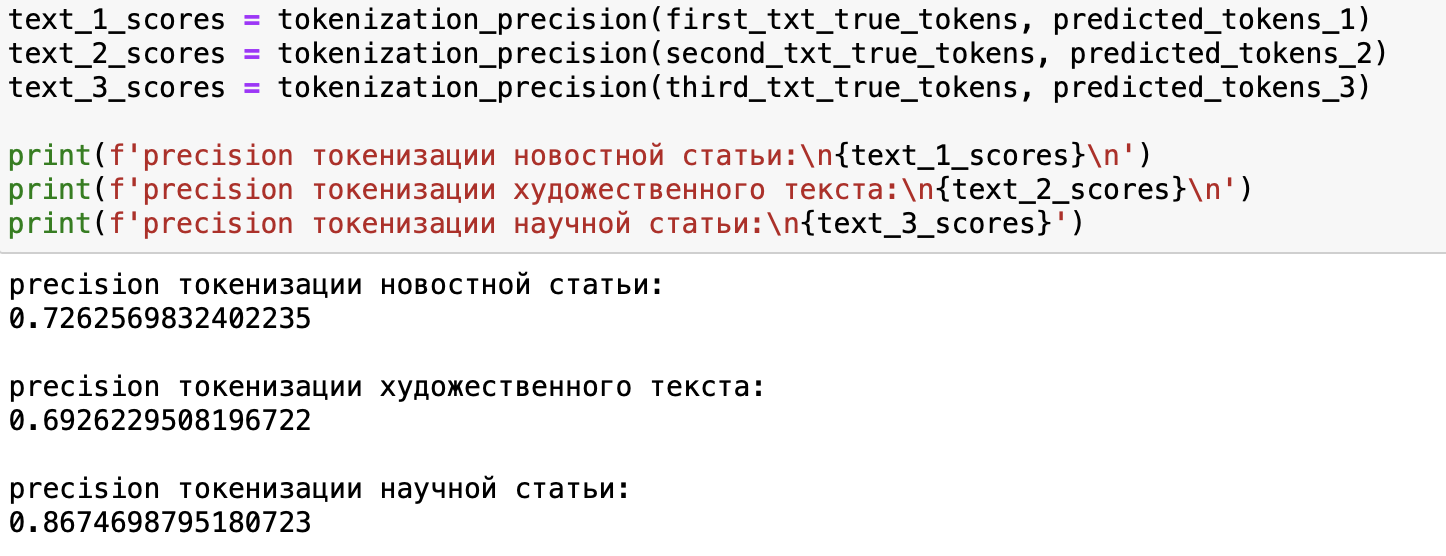
- Художественная литература: "Война и мир" Л.Н. Толстого.

Для каждого текста нужно подготовить эталонное разбиение на предложения и токены, а затем сравнить его с результатами, полученными с помощью функций split\_sentences и tokenize\_text.

Для оценки точности можно использовать метрику precision, которая показывают соотношение количества правильно определенных токенов или предложений к общему количеству токенов или предложений в эталонном разбиении.

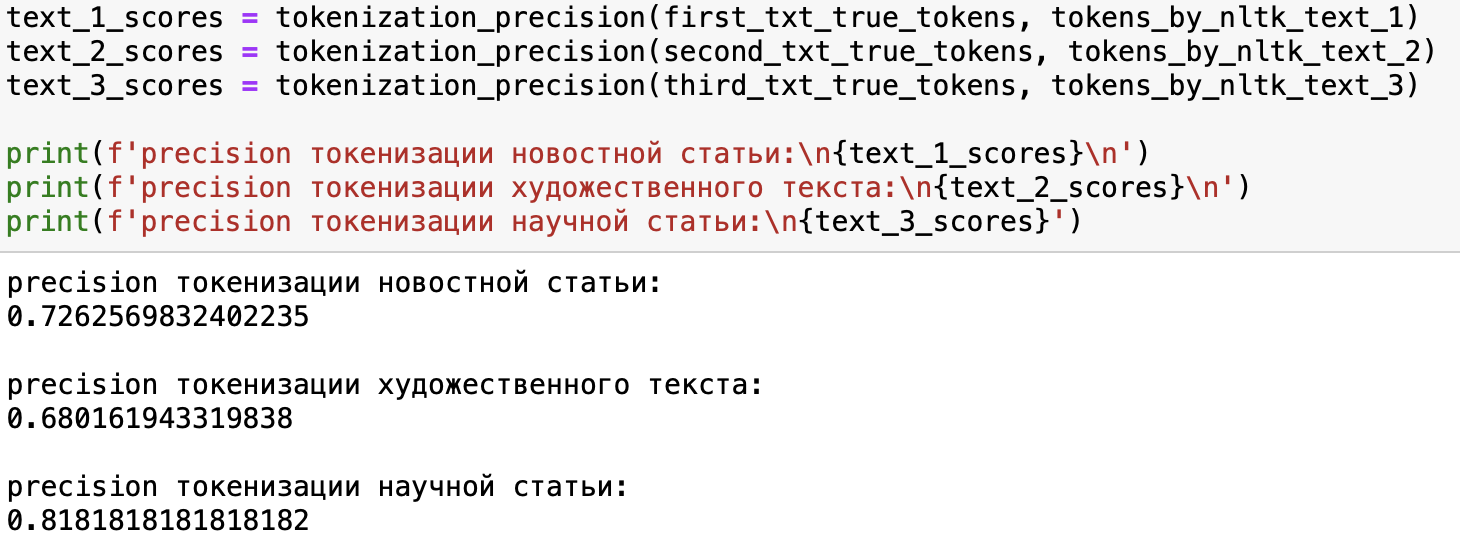


С помощью функции «open» читаем тексты и оцениваем точность работы функций на выбранных текстах:



Результат показывает, что лучше всего наши функции справляются с токенизацией научной статьи, а хуже всего с художественным текстом. Следует заметить, что использованные тексты имеют приблизительно одинаковый объем.

Теперь сравним результат работы функций с готовой функцией токенизации из библиотеки «nltk».



Результат оказался близким, хотя и несколько хуже. Более подробно с кодом можно ознакомиться в файле «tokenization.ipynb».