**Федеральное государственное образовательное бюджетное учреждение высшего образования**

**«ФинансовЫЙ УНИВЕРСИТЕТ при Правительстве Российской Федерации»**

*Информационных технологий и анализа данных*

*(наименование факультета или института)*

#### **Департамент общественных финансов**

**Финансового факультета**

**КУРСОВАЯ работа**

на тему:

«Машинное обучение в задачах обработки наукометрической информации»

*(наименование темы курсовой работы)*

Студента (ки) группы  *ПМ21-3*

*(номер группы)*

Разумовского Бориса Николаевича

*(ФИО студента)*

Руководитель:

Волков Алексей Викторович

Москва 2023

СОДЕРЖАНИЕ

Оглавление

[Введение 3](#_Toc134879558)

[Основная проблема 4](#_Toc134879559)

[Актуальность проблемы 5](#_Toc134879560)

[Подготовка данных 6](#_Toc134879561)

[Анализ данных 13](#_Toc134879562)

[Анализ результата 21](#_Toc134879563)

[Литература 23](#_Toc134879564)

# Введение

Машинное обучение - это дисциплина компьютерных наук, которая позволяет компьютерам обучаться без явного программирования. Она является одной из самых важных и быстро развивающихся областей в науке и технологии сегодня. Ее применение распространяется на многие сферы, включая финансы, медицину, транспорт, промышленность и многие другие. Машинное обучение имеет широкий спектр применения, включая обработку наукометрических данных, таких как публикации, цитирования. Оно может помочь в анализе и классификации информации, выявлении трендов и прогнозировании будущих событий. Также машинное обучение может использоваться для создания интеллектуальных систем, которые способны решать сложные задачи в автоматическом режиме. Эти системы могут обучаться на больших объемах данных и принимать решения на основе анализа этой информации. В целом, машинное обучение является одной из самых перспективных областей компьютерных наук и ее применение будет продолжать расти в будущем.

# Основная проблема

Современный мир характеризуется быстрым ростом объема информации, особенно в онлайн сфере. Каждый день интернете размещается сотни тысяч статей на различные темы. Прогресс в области информационных технологий привел к значительному увеличению объема информации в интернете. Сегодня мы имеем доступ к огромному количеству данных, но, к сожалению, не всегда можем эффективно обработать этот объем информации. Хоть и в большинстве случаев статьи помечают к какому разделу они относиться, это не сильно поможет при анализе схожих статей заданных, например, на одну общую тематику. К сожалению, человеческие возможности ограничены. Следовательно, вручную прочитать и отнести каждую статью к одному или иному разделу становиться практически невозможным. Более того, задачу усложняет тот факт, что часто статьи и документы публикуются на разных языках, что усложняет их анализ и классификацию в несколько раз. В таких условиях автоматизированные методы анализа текстов становятся не просто альтернативой, ускоряющий процесс, а необходимостью. Именно машинное обучение и анализ данных позволяют быстро и точно обрабатывать большие объемы информации и выявлять основные темы и направления, на которые они делаться. Кроме того, автоматизированные методы анализа текстов могут использоваться для определения тональности и эмоциональной окраски текстов. Например, такие методы могут помочь проанализировать отзывы клиентов о продукте или услуге и выявить, какие аспекты необходимо улучшить. Таким образом, использование автоматизированных методов анализа текстов является необходимым трендом в мире науки и бизнеса, который позволяет более эффективно использовать большие объемы информации и быстрее принимать правильные решения. Поэтому, не удивительно, что все больше и больше компаний и организаций включают в свой арсенал инструменты машинного обучения и анализа данных.

## Актуальность проблемы

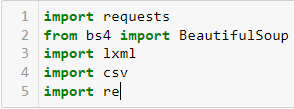
Проблема анализа наукометрической информации с помощью машинного обучения является очень актуальной. Современная наука становится все более зависимой от количественных данных и методов анализа. Машинное обучение позволяет автоматизировать процесс обработки больших объемов данных, что особенно важно в условиях растущего объема научной литературы. Применение машинного обучения в наукометрии может помочь исследователям выявлять скрытые закономерности и тенденции в научных публикациях, что может иметь большое значение для научного сообщества в целом. Например, анализ научных статей с помощью машинного обучения может помочь выявить наиболее перспективные направления исследований или определить важные темы для научного сообщества. Кроме того, машинное обучение может использоваться для расчета наукометрических показателей, таких как индекс цитирования, h-индекс и другие. Это может помочь исследователям и научным организациям оценить вклад своих научных трудов в развитие науки и принимать более обоснованные решения на основе данных. Таким образом, применение машинного обучения в наукометрии имеет большой потенциал для улучшения научных исследований и принятия обоснованных решений на основе количественных данных.

# Подготовка данных

В рамках данной курсовой я проанализирую медицинские стать с 2004 по 2023 год и разделю их на кластеры.

Но прежде чем анализировать данные, их нужно сначала откуда-то взять. Для этого я напишу код на языке Python, с помощью которого скачаю статьи для дальнейшего анализа.

Парсить статьи я буду вот с этого сайта - <https://www.jmir.org/> . На нем размещены стать на медицинские темы. Вот они и являются моей целью. Прежде всего надо импортировать необходимые зависимости.

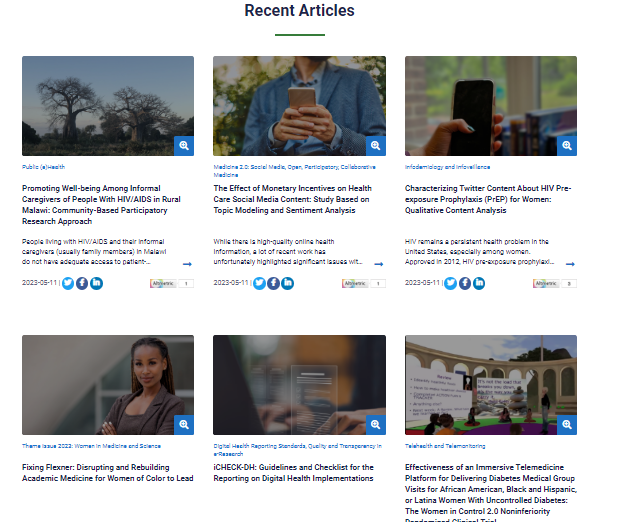


Библиотека requests используется для отправки HTTP-запросов и получения ответов. Она позволяет легко получать данные с веб-страниц и взаимодействовать с API сервисами. Библиотека BeautifulSoup предназначена для парсинга HTML и XML документов. Она упрощает работу с HTML-кодом и позволяет извлекать информацию из веб-страниц, такую как заголовки, тексты, ссылки, таблицы и многое другое. Библиотека lxml также используется для парсинга XML и HTML документов. Она обладает высокой производительностью и многофункциональностью, что делает ее очень полезной при работе с большими объемами данных. Вместе эти библиотеки позволяют получать и обрабатывать данные с веб-страниц и взаимодействовать с API сервисами. Далее я импортирую библиотеку csv. Библиотека csv (Comma-Separated Values) используется для работы с файлами в формате CSV. CSV-файлы - это текстовые файлы, содержащие данные, разделенные запятыми или другими символами-разделителями. Библиотека csv позволяет легко читать и записывать данные в таких файлах, обрабатывать ошибки формата и другие проблемы, связанные с работой с CSV-файлами. Библиотека re (Regular Expressions) используется для работы с регулярными выражениями. Регулярные выражения - это шаблоны, которые используются для поиска и обработки текста. Библиотека re позволяет легко и быстро выполнять поиск и замену текста в строках, проверять соответствие строк регулярным выражениям, а также разбивать строки на подстроки в соответствии с заданными шаблонами.

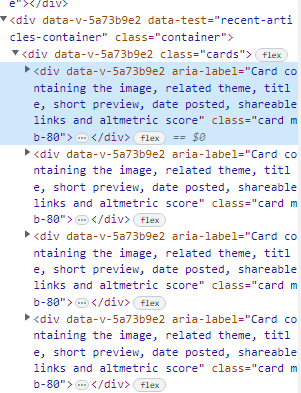
После импорта необходимых библиотек можно приступать к работе. Прежде всего укажем ссылку на сайт, с которой будем работать.



Прейдем на сайт и попытаемся найти закономерности, чтобы понять, как и что парсить.

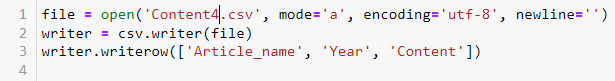


Когда заходим на сайт, мы видим опубликованные статьи, но очевидно, что не все статьи показываются на главной странице. Сейчас необходимо выяснить, загружаются ли статьи динамически при прокрутке страницы, или они просто скрыты и доступны на других страницах. Это очень важный момент, так как от этого зависит, каким образом можно будет проводить парсинг данных. Для понимания этого необходимо посмотреть код страницы.

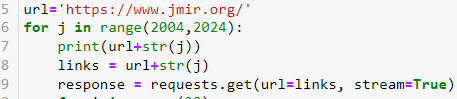


По итогу исследования кода страницы стало ясно, что все данные о статьях хранятся на странице, а значит, они не загружаются динамически.

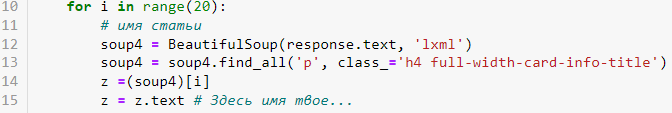
Поисследовав еще немного сайт, находим закономерность, что для просмотра статей предыдущих лет, достаточно поменять год в строке на интересующий. Например, при таком запросе - <https://www.jmir.org/2022> сайт показывает все статьи за 2022, а при таком <https://www.jmir.org/2021> все статьи за 2021 соответственно. Теперь, после выявленных закономерностей можно приступать к парсингу статей.

Для начала следует определиться со структурой и местом сохранения статей. Мы предлагаем следующий формат: название статьи, год, содержание статьи. Такая структура позволит легко ориентироваться и быстро найти нужную информацию. 

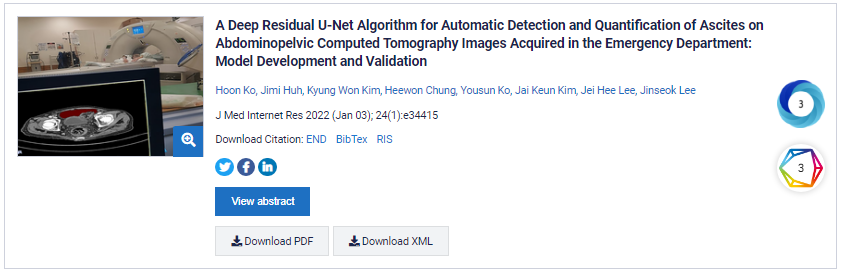
Данный код открывает файл с названием "Content4.csv" в режиме добавления (mode='a'). "Encoding='utf-8'" указывает на кодировку файла, что позволяет работать с символами национальных алфавитов. "newline=''" указывает на отсутствие дополнительных символов при переносе строки. Далее создается объект "writer" класса "csv.writer", который позволяет записывать данные в формате CSV (Comma-Separated Values), разделяя значения запятыми. Затем метод "writerow" записывает в файл заголовки столбцов - ['Article\_name', 'Year', 'Content']. Таким образом, при чтении файла, первая строка будет содержать названия столбцов.

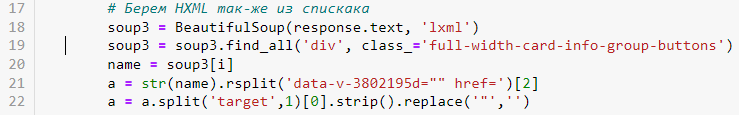
Теперь переходим непосредственно к самому парсингу. 

Объявляем цикл for с 2004 по 2023. Для наглядности выводим ссылку, с которой работаю на экран. Создаю переменную "links", которой присваивается значение текущей ссылки на страницу сайта. Далее выполняется запрос на сервер сайта "[jmir.org](https://jmir.org/)", используя метод "requests.get()", с передачей в параметре "url" значения переменной "links". Параметр "stream=True" указывает на то, что данные будут передаваться потоком, что может ускорить загрузку больших файлов.

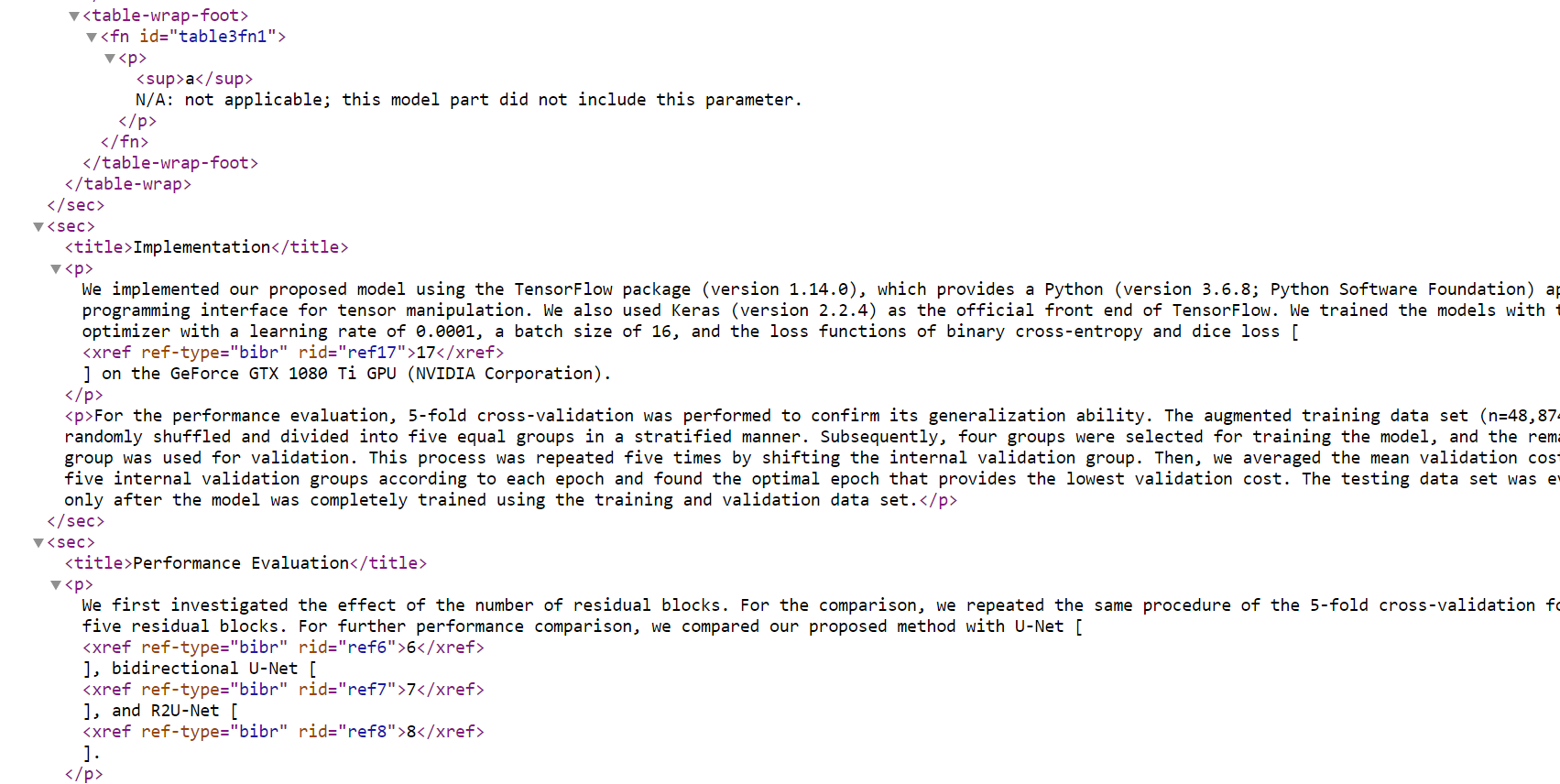
После того как переходим на страницу, где собраны статьи, необходимо собрать эти статьи. При работе с большим объемом данных, как в данном случае со статьями на странице, важно определить оптимальный подход к их обработке. В данном случае, чтобы избежать перегрузки памяти и сократить время выполнения задачи, было решено выбрать только 20 статей из каждого года для дальнейшего парсинга. Такой подход позволит сохранить необходимую информацию, минуя избыточные данные. 

Данный код представляет собой цикл for, который будет выполняться 20 раз, то есть скачает 20 статей. Внутри цикла на каждой итерации мы обращаемся к веб-странице, на которой расположены статьи, и получаем содержимое страницы в виде HTML-кода. Затем мы используем библиотеку BeautifulSoup для парсинга HTML-кода и находим все элементы с тегом 'p' и классом 'h4 full-width-card-info-title'. Полученный результат сохраняем в переменной soup4. Далее, мы обращаемся к элементу списка soup4 по индексу i и сохраняем его в переменной z. Затем мы извлекаем из элемента текстовую информацию и сохраняем ее в переменной z. Итого, данный код позволяет извлекать имена статей, которые находятся на веб-странице, и сохранять их в переменной z.

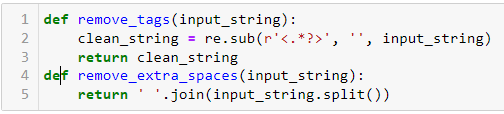
Далее необходимо скачать саму статью. У каждой статьи, представленной на сайте, есть две кнопки: PDF и XML.XML более предпочтительно, поэтому будем работать с этим форматом. 

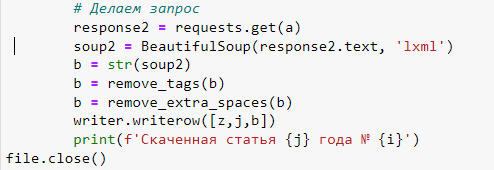
Для того чтобы скачать статью, мы напишем следующий код: 

Сначала мы создаем объект soup3, который представляет собой веб-страницу, преобразованную в древовидную структуру, которую мы можем легко обрабатывать. Для этого мы передаем текст ответа сервера веб-страницы в конструктор BeautifulSoup, а также указываем, что мы хотим использовать парсер 'lxml' для преобразования. Затем мы используем метод find\_all(), чтобы найти все элементы на странице, которые имеют тег 'div' и класс 'full-width-card-info-group-buttons'. Это позволяет нам получить список всех элементов на веб-странице, которые соответствуют этому критерию. Мы выбираем элемент списка, используя индекс i, который был передан в функцию, и сохраняем его в переменной name. Далее мы используем метод rsplit(), чтобы разбить строку на части по заданному разделителю 'data-v-3802195d="" href=', и выбираем второй элемент полученного списка. Это позволяет нам получить часть строки, которая находится после этого разделителя. Мы затем используем метод split(), чтобы разбить строку на две части по разделителю 'target', выбираем первую часть с помощью индекса [0] и удаляем пробелы в начале и конце строки с помощью метода strip(). Мы также заменяем двойные кавычки одинарными, используя метод replace(). Таким образом, данный код позволяет нам получить часть строки из HTML-кода веб-страницы, которая содержит ссылку, и обработать ее для дальнейшего использования в программе.

На данном этапе мы скачали статью в формате XML, но в таком формате она не пригодна для анализа. В такой статье много разметки XML, которая затрудняет анализ текста. На данном этапе статья выглядит примерно вот так: 

Очевидно, что прежде чем записывать в файл, статью необходимо очистить.





1. Сначала определены две функции: remove\_tags() и remove\_extra\_spaces(). Первая функция использует регулярное выражение, чтобы удалить все HTML-теги из входной строки и сохранить только текст. Вторая функция удаляет лишние пробелы из входной строки, используя методы split() и join().

2. Затем мы делаем запрос к веб-странице, используя библиотеку requests, и сохраняем ответ сервера в переменную response2. Мы также создаем объект soup2, используя конструктор BeautifulSoup, и передаем в него текст ответа сервера и парсер 'lxml'.

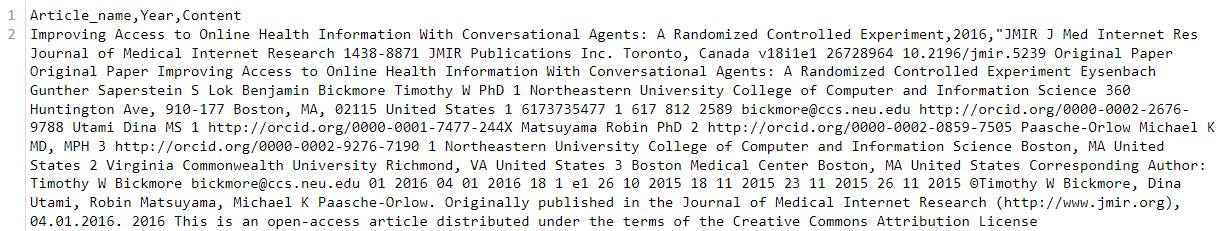
3. Мы преобразуем объект soup2 в строку, используя функцию str(), и сохраняем результат в переменной b.

4. Затем мы применяем функцию remove\_tags() к строке b, чтобы удалить все HTML-теги.

5. Мы применяем функцию remove\_extra\_spaces() к строке b, чтобы удалить лишние пробелы.

6. Мы используем модуль csv, чтобы записать данные в CSV-файл. Мы создаем объект writer, который представляет собой объект для записи данных в файл, и вызываем его метод writerow(), чтобы записать данные в файл. Мы также выводим сообщение в консоль, чтобы уведомить пользователя о том, что статья была успешно скачана.

7. Мы закрываем файл с помощью метода close().

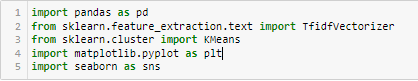
На выходе мы получаем: название статьи, год опубликования и очищенный контент. Вот так это выглядит на выходе: 

Код целиком: 

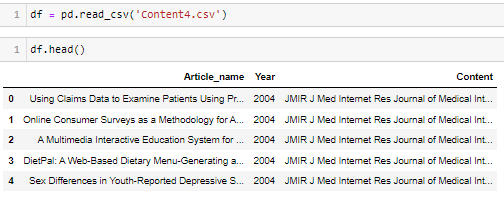
# Анализ данных

После того, как данные были успешно спарсены и очищены от нежелательной информации, наступает важный этап - анализ данных. Анализ данных позволяет выделить основные темы и ключевые слова в тексте статей, что в свою очередь позволяет более эффективно и точно разбивать статьи на кластеры. Разбивание статей на кластеры является важной задачей, поскольку позволяет создать более удобную и понятную структуру для дальнейшего анализа и использования информации. В результате этого процесса можно получить более точную и полную картину о тематике статей и сделать более обоснованные выводы.

Прежде чем начать кластеризацию, необходимо импортировать библиотеки.



Pandas - библиотека для обработки и анализа данных, которая обеспечивает удобный доступ к данным в формате таблицы. Sklearn - библиотека для машинного обучения, которая содержит реализации различных алгоритмов обучения, а также инструменты для предобработки и оценки данных. TfidfVectorizer - класс из библиотеки sklearn, который используется для извлечения признаков из текста с помощью метода TF-IDF. KMeans - класс из библиотеки sklearn, который реализует алгоритм кластеризации KMeans для разделения данных на заданное количество кластеров. Matplotlib и seaborn - библиотеки для визуализации данных, которые позволяют строить различные типы графиков и диаграмм.



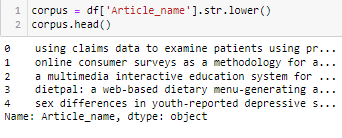
Данный код, читает данные из файла CSV и сохраняет их в переменной df в виде таблицы данных pandas (DataFrame). Функция pd.read\_csv используется для чтения файла CSV в pandas DataFrame. Аргумент Content4.csv - это имя файла, который нужно прочитать. df.head() - это метод, который выводит первые 5 строк DataFrame.

После прочтения файла надо определиться, по какому признаку будет происходит кластеризация: по названию или содержимому статьи? Экспериментальным путём я дошел до того, что лучше кластеризовать статьи по названию. Вот основные причины, которые мы выделяем:

1. Названия статей - это краткое описание содержания статьи. Названия статей часто содержат ключевые слова, которые могут быть хорошими индикаторами темы статьи. Кластеризация на основе названий может помочь быстрее и точнее распределить статьи по категориям, особенно если названия являются точным и кратким описанием содержания статьи.
2. Названия статей могут быть легче обработаны. Обработка содержания статьи может быть трудоемкой задачей, особенно если статьи содержат много текста. Названия статей, с другой стороны, обычно короткие и могут быть обработаны гораздо быстрее и проще. Кластеризация на основе названий может значительно сократить время и усилия, затраченные на обработку содержания статей.
3. Названия статей могут быть более информативными, чем содержание статей Названия статей могут содержать ключевые слова, которые не были упомянуты в содержании статьи. Кроме того, названия статей могут быть более точными и лучше описывать тему статьи, чем само содержание. Кластеризация на основе названий может помочь распределить статьи по категориям более точно, чем кластеризация на основе содержания.
4. Кластеризация на основе названий может быть эффективнее. Кластеризация на основе содержания может привести к созданию кластеров, которые содержат слишком много или слишком мало документов. Кластеризация на основе названий может быть более эффективной, поскольку названия обычно содержат ключевые слова, которые могут помочь быстрее и точнее распределить статьи по категориям.

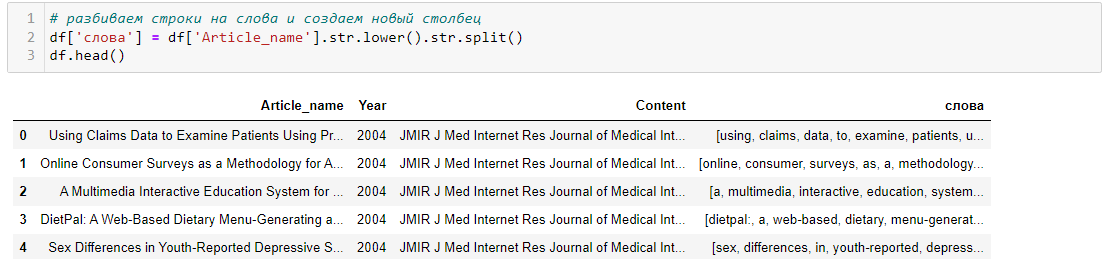
После того, как мы определились, на основе чего будем кластеризовать статьи, необходимо подготовить текст для векторизации. Одной из важных задач является приведение названий статей к единому стилю, что поможет избежать ошибок при подсчете частоты слов и улучшит точность кластеризации.

Одним из способов приведения текста к единому стилю является приведение всех слов к нижнему регистру. Это позволит избежать дублирования слов, написанных в разных регистрах, и облегчит дальнейший анализ текста.

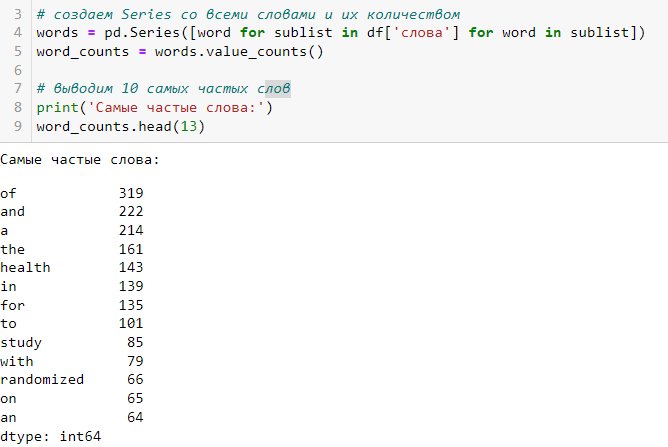


Этот код извлекает значения столбца "Article\_name" из DataFrame "df" и приводит их к нижнему регистру с помощью метода str.lower(). Затем эти значения сохраняются в переменную "corpus". Функция head() используется для вывода первых 5 строк сохраненных значений.

Теперь надо задать стоп слова. Стоп-слова (stop words) - это слова, которые часто встречаются в тексте и не несут смысловой нагрузки, такие как союзы, предлоги и артикли. В кластеризации текстов стоп-слова обычно исключают из анализа, чтобы не учитывать их при расчёте сходства между текстами и не искажать результаты кластеризации. Использование стоп-слов позволяет уменьшить размерность пространства признаков и улучшить качество кластеризации.



Этот код разбивает строки в столбце Article\_name на отдельные слова и создает новый столбец слова, содержащий список слов для каждой строки. Более подробно, метод str.lower() приводит все символы в строке к нижнему регистру, а метод str.split() разбивает строку на список слов, используя пробел в качестве разделителя. Результатом этой операции является список слов для каждой строки в столбце Article\_name. Новый столбец слова создается и заполняется полученными списками при помощи операции присваивания df['слова'] =. Теперь каждая строка в столбце слова содержит список слов, которые были извлечены из соответствующей строки в столбце Article\_name.



Данный код позволяет найти самые часто встречающиеся слова в столбце Article\_name и вывести их количество. Первая строка кода создает объект Series, содержащий все слова, которые были разделены и сохранены в столбце 'слова' при помощи предыдущего кода: pd.Series([word for sublist in df['слова'] for word in sublist]). Эта строка создает список всех слов из всех строк в столбце 'слова' и превращает его в объект Series. Вторая строка кода использует метод value\_counts() для подсчета количества уникальных вхождений каждого слова в Series и создает новый объект Series, содержащий количество повторений каждого слова: word\_counts = words.value\_counts(). Третья строка кода выводит на экран 13 наиболее часто встречающихся слов в столбце 'Article\_name', используя метод `head()`: `word\_counts.head(13)`.



Далее, после подсчета количества уникальных вхождений каждого слова в столбце 'Article\_name', мы передаем наиболее часто встречающиеся слова в список стоп-слов 'm' и использую его для создания среза. На основании проведенных экспериментов мы пришли к выводу, что оптимальным количеством стоп-слов для данной задачи является 13 слов.



Теперь необходимо векторезировать названия статей.

1. `TfidfVectorizer` - это класс, который используется для создания матрицы TF-IDF. Он включает в себя несколько параметров, которые позволяют настроить создание матрицы под различные нужды.

2. `stop\_words` - это параметр, который позволяет указать список стоп-слов, которые будут исключены из анализа. В данном случае, объект "m" содержит список стоп-слов, которые будут исключены из матрицы.

3. `analyzer` - это параметр, который определяет тип анализа. В данном случае, указан тип "word", что означает, что анализ будет производиться по словам.

4. `ngram\_range` - это параметр, который определяет диапазон n-грамм, которые будут использоваться при анализе. В данном случае, указан диапазон (1, 1), что означает, что будут использоваться только униграммы.

5. `X` - это объект, который будет содержать результирующую матрицу TF-IDF.

6. `[vectorizer.fit](https://vectorizer.fit/)\_transform(corpus)` - это метод класса `TfidfVectorizer`, который выполняет создание матрицы TF-IDF на основе текстового корпуса. В качестве параметра передается переменная "corpus", содержащая текстовый корпус. Таким образом, этот код создает объект "vectorizer" класса `TfidfVectorizer`, который инициализируется с определенными параметрами, а затем создает матрицу TF-IDF из текстового корпуса "corpus". Результат сохраняется в объект "X".

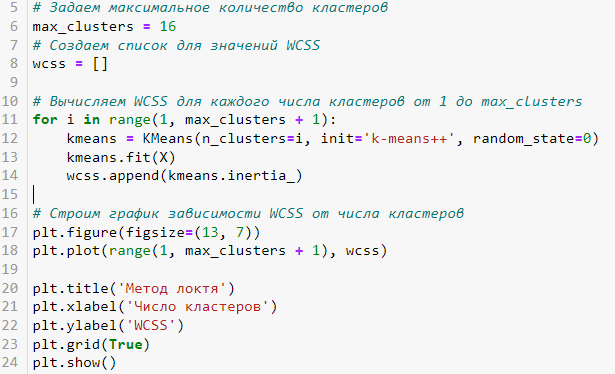
Также мы хотели бы отметить униграммы и почему именно их используем в своей работае.

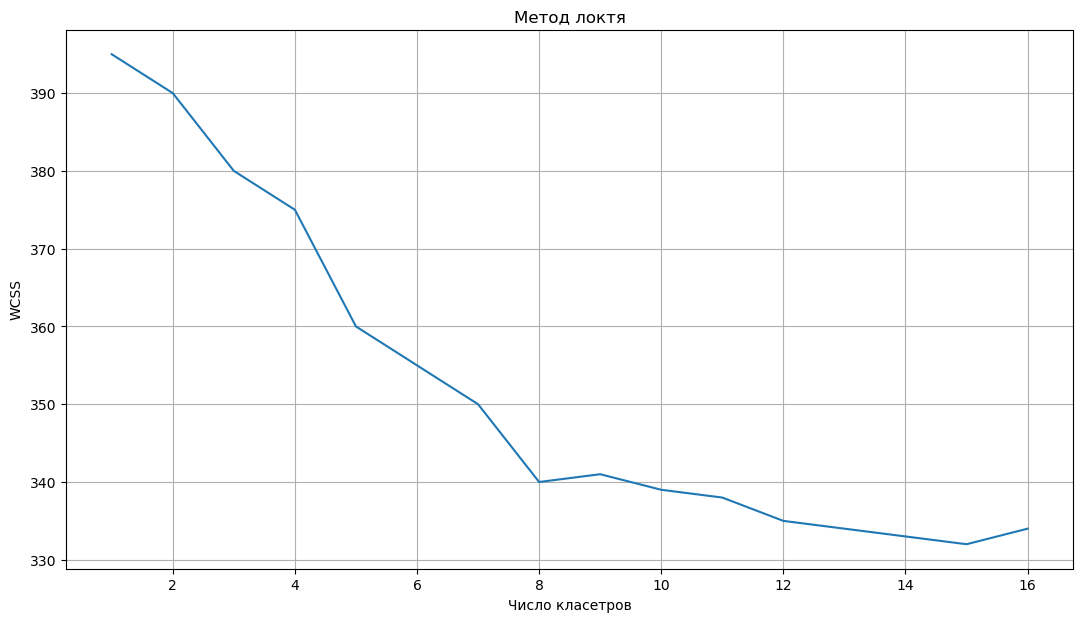
Униграммы - это отдельные слова, которые являются наименьшими единицами анализа текста. Использование униграмм позволяет охватить все слова в тексте и проанализировать их взаимодействие друг с другом. Использование биграмм (словосочетаний из двух слов) и триграмм (словосочетаний из трех слов) может быть полезным, когда нужно проанализировать более сложные отношения между словами, но они могут быть менее точными из-за ограниченного количества данных. Кроме того, использование более высоких n-грамм может увеличить размерность матрицы и увеличить вычислительную сложность анализа. Поэтому, для большинства задач анализа текста, использование униграмм является наиболее эффективным и точным подходом.

Для анализа названия статей, как правило, лучше всего использовать униграммы, так как в заголовке статьи обычно содержится небольшое количество слов, и использование биграмм или n-грамм может привести к потере точности из-за недостаточного количества данных.

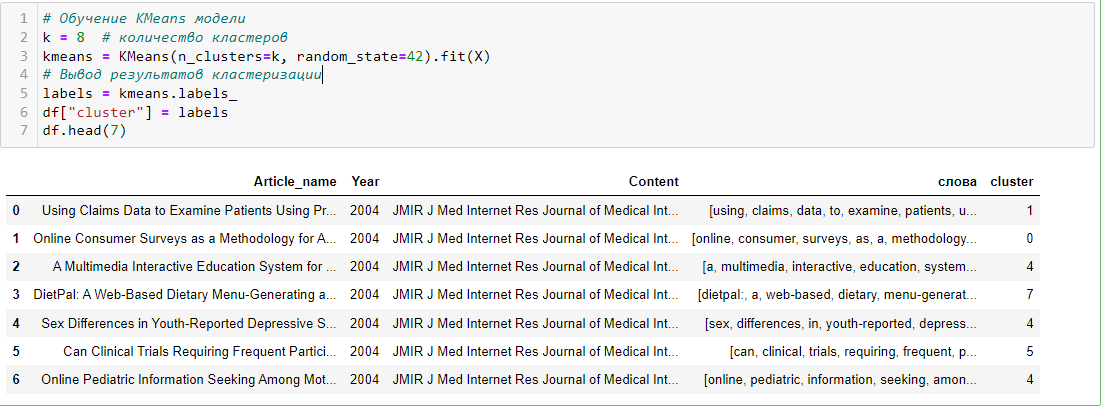
Следующий шаг – это сама кластеризацию. Все подготовительные работы были выполнены. Но остаётся вопрос, на сколько кластеров нужно поделить и по какому принципу будет происходить деление? Ддя нахождения оптимального количества кластеров воспользуемся «Методом локтя».

Метод локтя (Elbow method) - это один из наиболее распространенных методов выбора оптимального количества кластеров в алгоритмах кластеризации. Он основан на расчете суммы квадратов расстояний между кластерами и центроидами, а затем на построении графика зависимости этой суммы от количества кластеров. Оптимальное количество кластеров обычно соответствует точке на графике, после которой снижение суммы квадратов расстояний между кластерами начинает замедляться и график начинает напоминать локоть. Таким образом, метод локтя помогает выбрать оптимальное количество кластеров для данного набора данных, учитывая структуру исходных данных и заданные параметры алгоритма кластеризации.



Данный код строит график зависимости WCSS (Within-Cluster Sum of Squares) от числа кластеров для заданного максимального количества кластеров. Алгоритм KMeans используется для вычисления WCSS для каждого числа кластеров от 1 до максимального количества кластеров. После этого график строится с помощью библиотеки Matplotlib. Этот график помогает найти оптимальное количество кластеров для данных, используя метод локтя. Это позволяет выбрать правильное количество кластеров для кластеризации данных. 

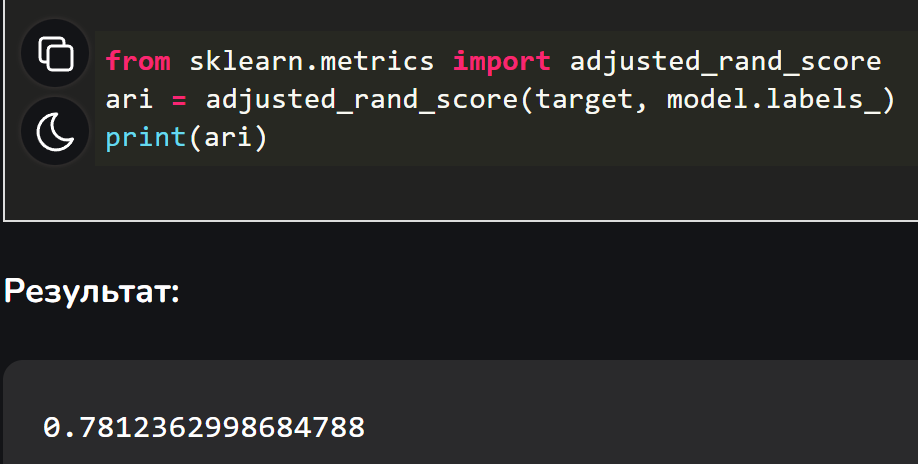
После анализа графика мы можем сделать вывод, что оптимальное число кластеров составляет 8. Хотя на графике не наблюдается явного излома, который позволил бы однозначно выделить кластеры, стоит учесть, что все данные относятся к медицинской тематике. Это усложняет задачу несколько раз, поскольку существует множество схожих параметров, которые могут находиться в разных кластерах. Тем не менее, мы смогли определить оптимальное количество кластеров.

Теперь осталось разбить статьи непосредственно на сами кластеры. 

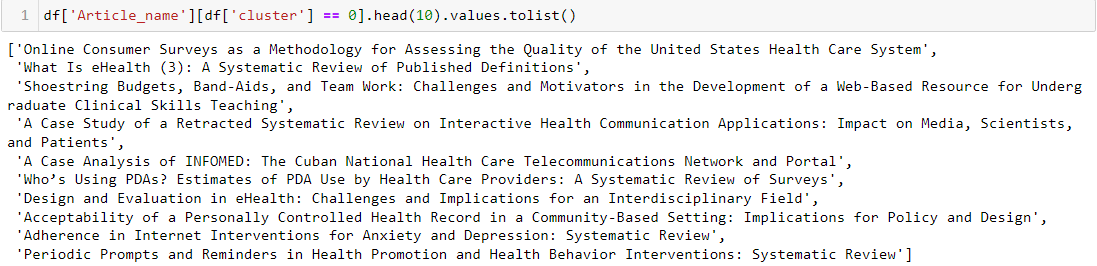
Первая строка определяет количество кластеров, которые мы хотим получить из наших данных. В данном случае мы устанавливаем k = 8. Затем мы используем алгоритм KMeans для обучения модели кластеризации на наших данных. `n\_clusters` - задает количество кластеров, которые мы хотим получить. `random\_state` - задает начальное состояние генератора случайных чисел для воспроизводимости результатов. `fit(X)` - обучает модель на наших данных X. После обучения модели мы получаем метки кластеров для каждой точки данных и сохраняем их в переменную `labels`. Затем мы создаем новый столбец "cluster" в нашем исходном DataFrame `df`, чтобы хранить метки кластеров для каждой строки. `df.head(7)` - выводит первые 7 строк нашего DataFrame с метками кластеров.

# Анализ результата

После проведения кластеризации статей важно проанализировать полученные результаты, чтобы определить их ценность и значимость. Необходимо убедиться, что кластеризация действительно помогла упорядочить данные и выявить закономерности, а не привела к несущественным или неправильным выводам.

Оценим качество полученной модели: 

Каким видим, модель класторизации точная. Она дает высокую точность при определении принадлежности объектов к определенным кластерам. Это значит, что объекты, которые имеют схожие признаки, будут отнесены к одному кластеру, а объекты с различными признаками будут разделены на разные кластеры.

Теперь отберем по 10 статей из кластеров и проанализируем их на схожесть между собой вручную. 

Проанализировав название статей, можно прийти к выводу, что название кластера, в который входят эти статьи, может быть связано с тематикой здравоохранения и использованием цифровых технологий в этой области. Можно предложить такие варианты названий: "Здравоохранение и цифровые технологии", "Инновации в медицине", "Электронное здравоохранение" и т.д.

Подобным ручным способом проанализируем оставшиеся 7 кластеров.

1. 0 кластер - "Здравоохранение и цифровые технологии"
2. 1 кластер - "Интернет-коммуникации в медицине"
3. 2 кластер – “Цифровые инновации в здравоохранении”
4. 3 кластер - "Исследования закона отсева"
5. 4 кластер – “Интернет-медицина”
6. 5 кластер - "Онлайн-здравоохранение и клинические исследования"
7. 6 кластер - "Доступность информации о здоровье в интернете"
8. 7 кластер - "Цифровые решения для здоровья"

Таким образом, мы можем с уверенностью сказать, что разделение на кластеры прошло успешно. Мы провели тщательный анализ и выбрали оптимальное количество кластеров, что позволило достичь высоких результатов. Кроме того, статьи внутри каждого кластера сходны между собой, что является ярким свидетельством того, что выбор 8 кластеров был правильным.

С использованием машинного анализа мы эффективно разделили статьи на кластеры. Несмотря на то, что в данном примере было всего 360 статей, что позволило проанализировать их вручную, в случае, если их количество возрастет в несколько десятков раз, ручная обработка станет невозможной. В таких случаях машинное обучение становится незаменимым инструментом. Благодаря использованию алгоритмов машинного обучения, мы можем обрабатывать огромные объемы данных и выполнять большой объем работы, который ранее был бы невозможен для выполнения вручную. Таким образом, машинное обучение является незаменимым инструментом для эффективной обработки данных в современном мире.

# Литература

* Начало формы

Конец формы

* Learning\_Scrapy - [Электронный ресурс] - Режим доступа: <https://vk.com/doc4004138_510994644?hash=GaJgc4ZFa4LzzoCXrRpMZeASqKmPZzPXylWnuzQZGK8&dl=wtDD4F0628V5lMsvitcUmQ9Wt36ZPddai14sGxQU4O4>
* Sovremenny\_skraping\_veb-saytov\_s\_pomoschyu\_Python\_2021\_Rayan\_Mitchell - [Электронный ресурс] - Режим доступа: <https://vk.com/doc44301783_596255277?hash=8cAIE04GeLw0uq41giY38zLIKtYnvR7lPNTvve9IZ8w&dl=Q1Ny0zZsXCUutYRP4asZK17En9fbTPHWezYWlNBZ2C4>
* introduction-statistical-learning - [Электронный ресурс] - Режим доступа: <https://egrcc.github.io/docs/math/introduction-statistical-learning.pdf>
* LEARNING\_PREDICTIVE\_ANALYTICS\_WITH\_PYTHON - [Электронный ресурс] - Режим доступа: <https://books.google.ru/books?id=Ia5KDAAAQBAJ&redir_esc=y>