ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**(ФИНАНСОВЫЙ УНИВЕРСИТЕТ)**

Кафедра анализа данных и машинного обучения

Факультета информационных технологий и анализа больших данных

***Дисциплина: «Машинное обучение в семантическом и сетевом анализе»***

*Направление подготовки: «Прикладная математика и информатика»*

*Профиль: «Анализ данных и принятие решений в экономике и финансах»*

*Факультет информационных технологий и анализа больших данных*

*Форма обучения очная*

*Учебный 2023/2024 год, 6 семестр*

**Курсовая работа на тему:**

«Построение согласованной базы знаний по предметной области на основе нескольких источников данных»

*Выполнил(а):*

студент(ка) группы ПМ21-2

Разумовский Борис Николаевич

*Научный руководитель:*

ассистент Блохин Н.В.

**Москва 2024**

Оглавление

[Ведение 3](#_Toc166448293)

[Глава 1. 5](#_Toc166448294)

[1.1 Изучение сайта 5](#_Toc166448295)

[1.2 Сбор ссылок на дела 6](#_Toc166448296)

[1.4 Создание Базы данных 7](#_Toc166448297)

[1.5 Заполнение Базы данных 8](#_Toc166448298)

[Глава 2. 10](#_Toc166448299)

[2.1 Извлечение юридических данных 10](#_Toc166448300)

[2.2 Поиск подхода к извлечению информации из документа 10](#_Toc166448301)

[2.3 Переформатирование документов 11](#_Toc166448302)

[2.4 Обновление Базы данных 14](#_Toc166448303)

[2.5 Парсинг документов 16](#_Toc166448304)

[2.6 Вставляем в данные в Базу Данных 19](#_Toc166448305)

[Глава 3. 21](#_Toc166448306)

[3.1 Зачем нужен граф? 21](#_Toc166448307)

[3.2 Построение графа 22](#_Toc166448308)

[Заключение 25](#_Toc166448309)

[Список использованных источников 27](#_Toc166448310)

[Полный код проекта 28](#_Toc166448311)

В**едение**

В современном мире данные становятся основным ресурсом для исследований, принятия решений и развития технологий в самых разных сферах. Объёмы информации растут экспоненциально, что создаёт как новые возможности, так и сложности в обработке и анализе данных. Одной из ключевых задач в этом контексте является построение согласованной базы знаний, которая позволяет интегрировать разрозненные данные из различных источников в единую, логически структурированную систему.

В ходе работы будет рассмотрено несколько ключевых аспектов: методы интеграции данных, алгоритмы обработки и анализа информации, а также технологии и инструменты, используемые для создания баз знаний. Особое внимание будет уделено вопросам качества данных, их актуальности и достоверности.

Актуальность данной темы высока в свете стремительного развития цифровых технологий и увеличения объемов данных в современном мире. Организации и предприятия сталкиваются с необходимостью эффективного управления и анализа информации для принятия обоснованных решений. Построение согласованной базы знаний становится ключевой задачей для обеспечения доступа к актуальным и достоверным данным из различных источников.

Цель исследования заключается в разработке методов интеграции данных, алгоритмов обработки и анализа информации, а также в оценке эффективности технологий и инструментов для создания баз знаний. Основной задачей является создание единой, логически структурированной системы, способной объединять разнообразные данные в целях улучшения принятия решений и развития технологий.

Предметом исследования являются методы интеграции данных, алгоритмы обработки и анализа информации, а также технологии и инструменты, используемые для создания баз знаний. Объектом исследования выступают данные различных типов и форматов, включая структурированные и неструктурированные данные, получаемые из различных источников.

##### Глава 1. Парсинг документов

## Изучение сайта

Для успешного парсинга сайта крайне важно понять, какие средства защиты использует сайт для предотвращения несанкционированного доступа. Например, наличие капчи, методы динамической загрузки контента и способы предоставления HTML-разметки могут значительно усложнить задачу.

В данной работе рассматривается парсинг сайта Московского городского суда (https://mos-gorsud.ru/). Этот сайт представляет собой унифицированную базу юридических дел по Москве, включая все категории дел — от административных до уголовных.

При попытке запросить данные с помощью обычных HTTP-запросов, сайт возвращает пустой ответ, что свидетельствует о сложной системе защиты от спама. Дополнительная сложность заключается в том, что информация о делах загружается динамически: веб-страница не содержит прямых ссылок в своей HTML-разметке, а данные о делах подгружаются в ответ на действия пользователя, например, при прокрутке страницы.

Также важно отметить, что на главной странице отображаются только ссылки на отдельные дела. Сами дела располагаются на подстраницах, куда ведут эти ссылки, и содержат всю необходимую информацию. Это означает, что для полного сбора данных необходимо разработать механизм, способный эмулировать действия пользователя и обрабатывать JavaScript, что значительно усложняет процесс парсинга.

Для преодоления технических сложностей, связанных с парсингом защищённых веб-сайтов, я использовал библиотеку Playwright для Python. Этот инструмент позволяет полноценно эмулировать действия браузера, включая выполнение JavaScript и взаимодействие с элементами веб-страницы, как это делает человек. Это значительно расширяет возможности сбора данных по сравнению с традиционными методами, такими как использование библиотек requests или Beautiful Soup, которые не могут исполнять JavaScript и обрабатывать динамические элементы страницы.

## Сбор ссылок на дела

В рамках моей курсовой работы, я решил применить автоматизированный подход к сбору данных с сайта Московского городского суда, используя для этой цели библиотеку Playwright. Этот инструмент предоставляет мощные возможности для эмуляции действий браузера, что позволяет эффективно обходить защитные механизмы сайтов и извлекать нужные данные.

Целью данного сценария является автоматизированный сбор ссылок на юридические дела, размещенные на указанном сайте. Сценарий начинается с инициализации сессии Playwright, в которой создается браузер в режиме без графического интерфейса (headless mode), что ускоряет его работу и уменьшает ресурсные требования.

Первым шагом я перехожу на страницу поиска сайта, вводя в поисковую строку название интересующей организации — в данном случае, "Сбербанк". Инициализируется первая страница результатов поиска, где я предусмотрительно делаю задержку в 25 секунд, чтобы убедиться, что весь динамически загружаемый контент был полностью подгружен.

Далее начинается процесс итеративного перелистывания страниц с результатами. С помощью цикла я методично обрабатываю каждую страницу: сначала извлекаю HTML-контент текущей страницы, затем передаю его в функцию `get\_links`, которая с помощью парсера BeautifulSoup находит и извлекает необходимые гиперссылки. Эти ссылки записываются в файл `datalinks.csv`, что позволяет аккумулировать результаты для последующего анализа.

Функция `get\_links` работает путем поиска элементов с классом `megasearch-result-item` — это контейнеры, содержащие информацию о каждом деле, включая нужные нам ссылки. Парсер ищет внутри каждого такого элемента тег `<a>`, содержащий атрибут `href`, который и является целевой ссылкой.

После обработки страницы сценарий кликает на кнопку "Следующая", чтобы перейти к следующей странице результатов, и весь процесс повторяется. Это продолжается до тех пор, пока не будут обработаны все предусмотренные страницы (в данном случае, 676).

В завершение работы скрипта, контекст браузера и сам браузер закрываются, что является важным шагом для освобождения ресурсов системы.

Этот подход позволяет мне эффективно собирать большое количество данных в автоматизированном режиме, что значительно ускоряет процесс подготовки курсовой работы и повышает качество исследования за счет увеличения объема обрабатываемой информации.

## Создание Базы данных

В данном разделе курсовой работы, я перехожу к следующему ключевому этапу — созданию базы данных, которая будет использоваться для систематического хранения собранных данных. Использование базы данных позволяет эффективно организовать информацию, обеспечить быстрый доступ к данным и их анализ. В качестве системы управления базой данных (СУБД) я выбрал SQLite — легковесную, файловую СУБД, которая идеально подходит для целей научных исследований и разработки прототипов без необходимости сложной настройки сервера баз данных.

Процесс начинается с создания файла базы данных legal\_cases.db, который будет использоваться для хранения всех собранных данных. С помощью языка программирования Python и библиотеки sqlite3, я инициирую создание таблицы в базе данных. Таблица, названная collection, предназначена для систематизации информации о каждом судебном деле. Структура таблицы разработана таким образом, чтобы отражать ключевые атрибуты дел, такие как уникальный идентификатор дела, номер дела, истец, ответчик, заявитель, судья, статья КоАП РФ, суд, вынесший решение, категория дела, текущее состояние, результат рассмотрения, ссылка на документ, и связанные законы.

Создание таблицы осуществляется с помощью SQL-запроса, который проверяет, существует ли уже такая таблица, и, если нет — создает ее. Это обеспечивает гибкость и удобство при многократных запусках скрипта, так как исключает возможность возникновения ошибок из-за попыток создать уже существующую таблицу.

Данный подход позволяет не только эффективно управлять большим объемом данных, но и обеспечивает возможность выполнения сложных запросов к данным, анализа и обработки информации в дальнейшем. Таким образом, база данных становится ключевым инструментом в моей курсовой работе, позволяющим глубоко анализировать судебную практику и тенденции в разрешении юридических дел на территории города Москвы.

## Заполнение Базы данных

В рамках курсовой работы по автоматизированному сбору и анализу данных, я разработал асинхронную систему для эффективного извлечения информации с веб-сайта Московского городского суда. Этот проект требует обработки значительного объема данных, включая судебные решения и юридические документы, что делает важным использование асинхронных методов для оптимизации процесса.

Процесс начинается с загрузки списка ссылок из файла `datalinks.csv`, содержащего URL-адреса страниц с информацией о конкретных судебных делах. Затем, с использованием библиотеки Playwright, система асинхронно обращается к каждой из этих ссылок. Playwright позволяет не только загружать страницы, как это делает обычный браузер, но и выполнять сложные взаимодействия с веб-страницами, такие как навигация по ссылкам или скачивание файлов, что исключительно важно для комплексной автоматизации задач.

Каждая страница, загружаемая в процессе работы программы, подвергается тщательному анализу с целью извлечения максимального количества полезной информации. С помощью библиотеки BeautifulSoup программа анализирует HTML-структуру страницы и извлекает такие важные данные, как уникальные идентификаторы дел, номера дел и жалоб, а также информацию о сторонах конфликта: истце, ответчике и заявителе. Эта информация критична для создания полной картины судебного разбирательства и помогает систематизировать данные для дальнейшего анализа.

Для каждого дела также извлекаются ссылки на документы, которые затем скачиваются и сохраняются локально. Путь к сохраненному файлу, вместе с остальной извлеченной информацией, записывается в базу данных SQLite. Использование асинхронной базы данных aiosqlite обеспечивает быструю и надежную обработку данных без блокировки основного потока выполнения программы.

Семафоры в асинхронной программе используются для ограничения количества одновременно выполняемых задач, что предотвращает перегрузку сервера и повышает надежность процесса сбора данных. В данном случае, система ограничивает количество одновременно открытых веб-страниц, что позволяет избежать чрезмерного потребления ресурсов и потенциальных ошибок из-за исчерпания системных ресурсов.

В процессе работы активно используется логирование для отслеживания хода выполнения программы и быстрой диагностики возможных проблем. Логгер фиксирует все ключевые этапы обработки данных, включая ошибки при загрузке страниц или в процессе извлечения данных. Это позволяет быстро реагировать на возникающие проблемы и обеспечивает возможность корректировки процесса в реальном времени.

##### Глава 2. Обработка документов

## Извлечение юридических данных

После успешной загрузки юридических документов, связанных с различными судебными делами, передо мной стоит новая задача — обработка и анализ содержащихся в них данных. Основной интерес представляет информация о законах и кодексах, согласно которым были вынесены судебные решения. Среди наиболее значимых для анализа документов — Гражданский кодекс и Административный кодекс, а также другие правовые акты, которые играют ключевую роль в правоприменительной практике.

Одним из основных вызовов при обработке скачанных документов является их неоднородная текстовая структура. Документы могут быть оформлены в различных форматах и стилях, что делает процесс автоматического извлечения данных сложным и требующим особого подхода. Например, одни документы могут содержать чётко оформленные разделы и подразделы, в то время как другие — непрерывный текст, где законы и кодексы упоминаются спорадически и без явной структурированности.

## Поиск подхода к извлечению информации из документа

В ходе разработки проекта по обработке юридических документов, возникла необходимость применения технологий искусственного интеллекта для извлечения данных из текстов без четкой структуры. Изначально рассматривались два основных подхода к использованию технологий Large Language Models (LLM): самостоятельное дообучение предварительно обученной модели, доступной на платформе Hugging Face, или использование модели через API, предоставляемое различными сервисами.

Первый вариант предполагал загрузку и последующее дообучение одной из доступных моделей. Это потребовало бы значительных вычислительных ресурсов, включая специализированное оборудование, такое как GPU или TPU, для эффективной обработки данных. Кроме того, этот процесс включает в себя подготовку обучающего датасета, настройку параметров обучения, а также мониторинг процесса обучения, что представляет собой трудоемкую и потенциально дорогостоящую задачу. В силу сложности и стоимости реализации, данный подход был признан нецелесообразным для нашего проекта на данном этапе.

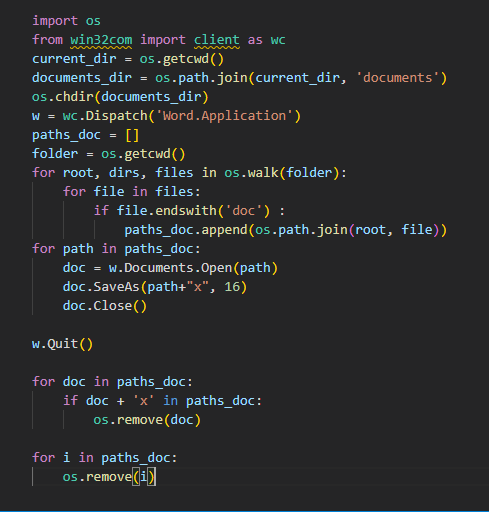
Второй вариант заключался в использовании готовых моделей через API, что является более простым и экономически выгодным решением. Этот подход позволяет избежать затрат на дообучение и обслуживание моделей, так как сервисы предоставляют доступ к уже обученным и оптимизированным моделям. Однако, в ходе тестирования бесплатных версий таких API выяснилось, что они не справляются с задачей извлечения данных из сложных юридических текстов. Проблемы возникали из-за ограничений на количество запросов, время обработки и, что наиболее критично, из-за недостаточной специализации моделей на юридическом контенте. В результате, модели демонстрировали низкую точность в распознавании и классификации юридически значимых элементов текста.

Оценив оба варианта, было принято решение отказаться от использования LLM для данного проекта и искать альтернативные методы. Этим альтернативным методом оказался паркинг документа через старую, проверенную временем, безотказную библиотеку re (regular expression).

## Переформатирование документов

После выбора метода парсинга текстовых данных с использованием регулярных выражений из библиотеки `re`, возникла необходимость стандартизировать форматы всех документов для обеспечения более эффективной обработки. Большинство существующих документов находилось в формате `.doc`, который, хоть и широко распространён, представляет определённые трудности при работе с современными Python-библиотеками из-за его устаревшей структуры. Следовательно, было принято решение конвертировать эти файлы в более современный и удобный для обработки формат `.docx`.

Для выполнения конвертации я использовал библиотеку `pypiwin32`, которая позволяет взаимодействовать с приложениями Microsoft Office через COM-интерфейс непосредственно из Python. Процесс начинается с настройки рабочего окружения:



Фрагмент кода из файла helper.ipynb

1. Инициализация рабочей директории:

- Скрипт начинает с определения текущего рабочего каталога, используя функцию `os.getcwd()`.

- Затем формируется путь к папке `documents`, расположенной в текущем каталоге, что становится новым рабочим каталогом для скрипта.

2. Поиск файлов для конвертации:

- Скрипт выполняет обход всех файлов в указанной директории и её поддиректориях, используя `os.walk()`, и собирает пути к файлам с расширением `.doc`.

3. Конвертация файлов:

- Для каждого найденного файла создаётся объект документа Word с помощью метода `Dispatch('Word.Application')` из библиотеки `win32com`.

- Каждый документ открывается, сохраняется в формате `.docx` с добавлением 'x' к существующему имени файла, и затем закрывается.

4. Очистка:

- После конвертации исходные файлы `.doc` удаляются для освобождения дискового пространства и избежания дублирования данных.

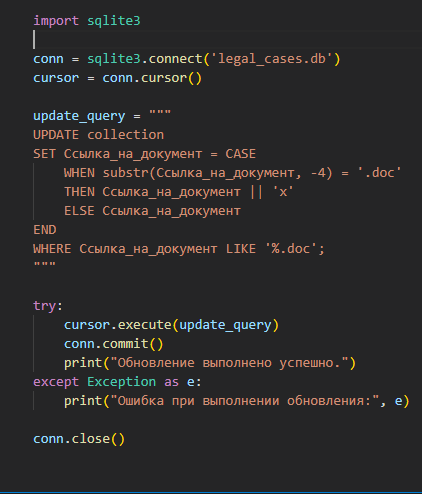
По завершении процесса конвертации, Word полностью закрывается с помощью метода `Quit()`, что гарантирует корректное освобождение системных ресурсов. Все действия логируются, включая изменения в рабочем каталоге и статусы конвертации, что обеспечивает прозрачность процесса и возможность отладки в случае возникновения ошибок.

Этот метод конвертации файлов является ключевым шагом в подготовке данных к дальнейшему анализу и обработке, обеспечивая не только единообразие форматов, но и совместимость с современными инструментами программирования и анализа данных.

## Обновление Базы данных

Перед началом процесса парсинга данных из документов, конвертированных в формат `.docx`, необходимо выполнить важную операцию — обновление ссылок на документы в базе данных. После конвертации файлов из формата `.doc` в `.docx`, ссылки на эти файлы, хранящиеся в базе данных, остались в старом формате, что может привести к ошибкам при попытках доступа к файлам. Следовательно, требуется корректировка путей в базе данных для обеспечения их актуальности и корректности.

Для обновления ссылок на документы в базе данных используется язык SQL в контексте Python-библиотеки `sqlite3`, что позволяет взаимодействовать с локальной SQLite базой данных.



Фргамент кода из файла helper.ipynb

1. Подключение к базе данных:

- С помощью функции `connect()` устанавливается соединение с файлом базы данных `legal\_cases.db`. Это действие возвращает объект соединения.

- Далее создается объект курсора с помощью метода `cursor()`. Курсор используется для выполнения SQL-запросов.

2. Формирование SQL-запроса для обновления данных:

- Запрос строится таким образом, чтобы проверить каждую запись в столбце `Ссылка\_на\_документ` таблицы `collection`.

- С помощью функции `substr()` проверяется, заканчивается ли текущая ссылка на документ расширением `.doc`. Если условие выполняется, то к текущему значению ссылки добавляется символ 'x', тем самым преобразуя расширение в `.docx`.

- Обновление происходит только для тех строк, где ссылка на документ содержит `.doc`.

3. Выполнение SQL-запроса:

- Запрос выполняется методом `execute()`, после чего изменения фиксируются в базе данных методом `commit()`.

- В случае успешного выполнения запроса выводится сообщение об успешном обновлении.

- Если в процессе выполнения запроса произошла ошибка, она перехватывается в блоке `except` и выводится сообщение об ошибке.

4. Закрытие соединения с базой данных:

- После выполнения обновления соединение с базой данных закрывается методом `close()`, чтобы освободить системные ресурсы и предотвратить утечку данных.

Этот шаг обновления ссылок критически важен для поддержания целостности данных и обеспечения бесперебойной работы последующих этапов проекта, связанных с анализом содержимого документов.

## Парсинг документов

Для извлечения юридически значимой информации из конвертированных документов `.docx`, был разработан эффективный метод парсинга, основанный на использовании регулярных выражений. Чтобы повысить точность и полноту извлечения данных, был создан обширный список регулярных выражений, ориентированных на различные юридические термины и форматы ссылок на законодательные акты.



Фрагмент кода из файла helper.ipynb

1. Подключение к базе данных:

Инициализация подключения к базе данных SQLite осуществляется для доступа к информации о расположении документов. С помощью SQL-запроса выбираются идентификаторы документов и их пути из таблицы `collection`.

2. Извлечение путей к документам:

Выполнение SQL-запроса и сохранение результатов позволяет получить список путей к документам, которые необходимо обработать.

3. Парсинг документов:

Функция `parser\_doc()` отвечает за обработку каждого документа. В ней документ открывается и анализируется на предмет наличия юридических ссылок с использованием списка регулярных выражений.

Работа функции `parser\_doc()`:

- Открытие документа: Используя библиотеку `docx`, документ открывается по указанному пути.

- Сбор текста: Весь текст документа собирается в единую строку, что позволяет эффективнее применять регулярные выражения.

- Применение регулярных выражений: Для каждого выражения в списке `patterns` происходит поиск соответствий в тексте документа. Выражения специально разработаны для поиска ссылок на законы РФ, статьи Гражданского и Гражданско-процессуального кодексов РФ.

- Категоризация результатов: Найденные соответствия категоризируются и сохраняются в словаре `found\_laws` по типам документов, таким как законы, статьи ГК РФ и ГПК РФ.

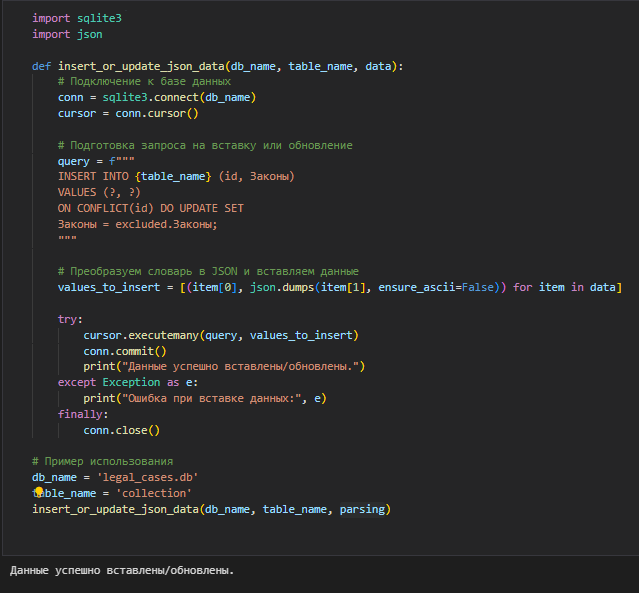
В функции предусмотрена обработка исключений, которая позволяет избежать сбоев в случае возникновения ошибок при чтении документов. В случае ошибки функция возвращает идентификатор документа и пустой словарь, что позволяет легко идентифицировать документы, требующие дополнительной проверки.

Этот подход к парсингу документов обеспечивает высокую степень автоматизации процесса извлечения данных и позволяет систематизировать информацию для последующего анализа и использования в юридических исследованиях. Каждый шаг процесса тщательно продуман, чтобы обеспечить максимальную точность и полноту извлеченных данных, а также для поддержания целостности и актуальности информационной базы.

В завершение процесса автоматизированного анализа юридических документов, программа успешно генерировала массив кортежей, каждый из которых представлял собой структурированный результат парсинга отдельного документа. Каждый кортеж содержал два элемента, отражающих результаты обработки: первый элемент — это уникальный идентификатор (**id**) каждого дела, хранящийся в базе данных, который обеспечивает возможность точной идентификации и связывания с соответствующим документом. Второй элемент кортежа — это словарь, который структурированно хранит все найденные в документе законы, разделённые по категориям.

## Вставляем в данные в Базу Данных

После успешного парсинга документов и извлечения нужной юридической информации, следующим критически важным шагом является интеграция этих данных в базу данных. Это позволит сохранить результаты для дальнейшего использования и анализа. Процесс интеграции включает в себя не только сохранение новых данных, но и обновление уже существующих записей в базе данных в случае необходимости.



Фрагмент кода из файла helper.ipynb

Функция `insert\_or\_update\_json\_data` создана для того, чтобы автоматизировать процесс вставки или обновления информации в базе данных. Эта функция принимает три параметра: имя базы данных (`db\_name`), имя таблицы (`table\_name`) и данные (`data`), которые необходимо вставить или обновить.

С помощью библиотеки `sqlite3` устанавливается соединение с базой данных, указанной в параметре `db\_name`. Соединение используется для создания курсора, который позволяет выполнять SQL-запросы.

SQL-запрос, подготовленный для выполнения операции вставки с возможностью обновления в случае конфликта, использует конструкцию `INSERT INTO ... ON CONFLICT ... DO UPDATE`. Это обеспечивает добавление новых данных, а при наличии записи с таким же ID происходит обновление поля `Законы`.

Перед выполнением запроса данные, представляющие собой словарь законов, преобразуются в формат JSON с помощью библиотеки `json`. Это позволяет сохранить структурированную информацию в одном поле базы данных. Для каждой записи формируется кортеж, содержащий ID и сериализованные в JSON данные. Эти кортежи вставляются или обновляются в базе данных с помощью метода `executemany`, который позволяет эффективно обработать множество записей одновременно.

В процессе вставки данных могут возникнуть ошибки, например, связанные с нарушением целостности данных или проблемами доступа к базе данных. Для обработки таких ситуаций используется блок `try-except`. При возникновении исключения выводится сообщение об ошибке. Независимо от результата выполнения запроса, соединение с базой данных закрывается для освобождения ресурсов.

Интеграция обработанных данных в базу данных является ключевым элементом процесса управления юридической информацией. Это не только обеспечивает сохранность данных, но и упрощает их последующий анализ и обеспечивает легкий доступ к ним для всех заинтересованных сторон.

##### Глава 3. **Построение графа**

## Зачем нужен граф?

Теперь, когда все необходимые данные успешно интегрированы и хранятся в базе данных, у нас образовалась согласованная и структурированная информационная система, содержащая данные из различных источников. Это создает идеальные условия для дальнейшего анализа и визуализации информации, что позволяет приступить к следующему значимому этапу проекта — построению графа взаимосвязей.

Использование графов в анализе данных представляет собой мощный инструмент, который позволяет визуализировать и анализировать сложные взаимосвязи между различными элементами данных. В контексте юридической информации, построение графа может помочь выявить связи между различными законами, статьями кодексов, судебными делами и другими юридически значимыми объектами. Такой подход позволяет:

- Определить ключевые законы и статьи, которые чаще всего используются в определенных категориях дел.

- Выявить возможные законодательные пробелы или пересечения, основываясь на частоте и контексте упоминаний определенных норм в различных делах.

- Анализировать тренды и изменения в законодательстве на основе динамики ссылок на законы и статьи во времени.

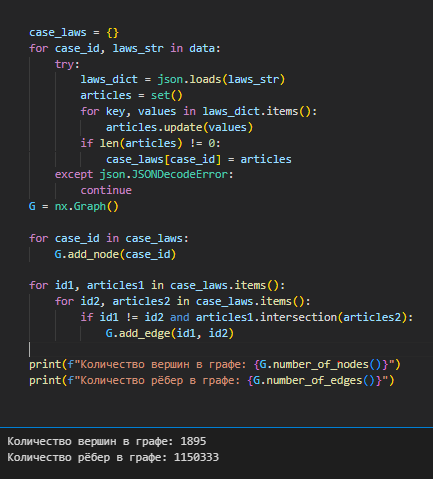
## Построение графа

Теперь, когда вся необходимая информация систематически организована и хранится в базе данных, предстоит выполнить важный этап анализа данных — построение графа взаимосвязей между судебными делами. В этом графе каждое дело будет представлено в виде вершины, а связи (рёбра) между ними будут формироваться на основе совместного использования одного или нескольких законов. Этот подход позволяет наглядно демонстрировать, как различные дела связаны через общее законодательство, и понять, какие законы являются наиболее значимыми в различных юридических контекстах.

Структура графа:

- Вершины графа: Каждая вершина в графе представляет собой отдельное юридическое дело. Информация о каждом деле, включая его уникальный идентификатор и ссылки на применённые законы, извлекается из базы данных.

- Рёбра графа: Рёбра между вершинами устанавливаются на основе пересечения в применении законов между делами. Если два дела ссылаются хотя бы на один общий закон, между их вершинами в графе проводится ребро.



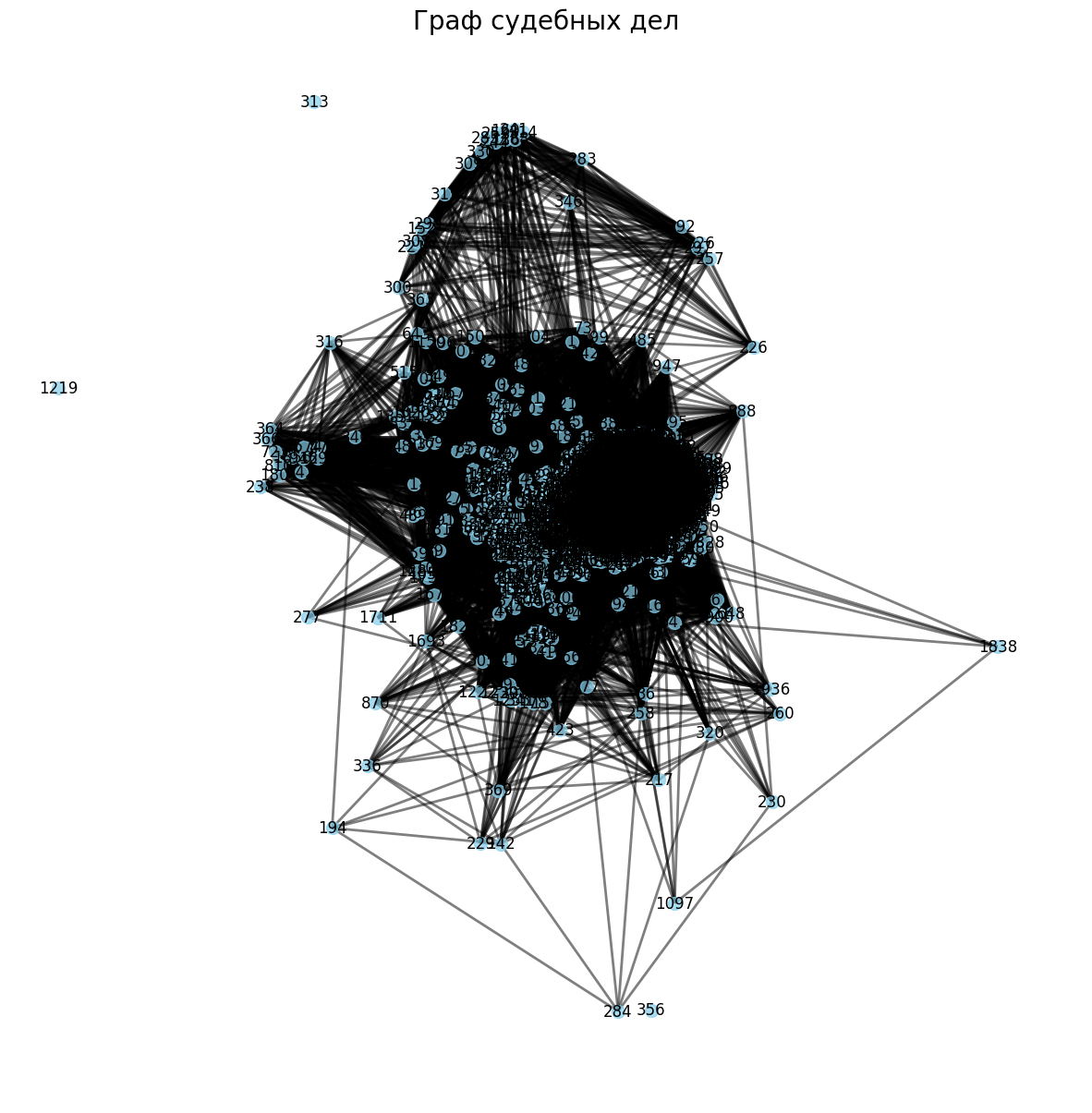
Фрагмент кода из файла create\_graph.ipynb

1. Из базы данных извлекаются записи о каждом судебном деле, включая идентификатор дела и связанные с ним законы. Эти законы представлены в формате JSON, который расшифровывается в структуру данных, позволяющую программно их обрабатывать. Затем из этой структуры формируется список уникальных статей законов для каждого дела.

2. На основе полученных данных создается граф, где каждое судебное дело представлено вершиной.

3. Для каждой пары дел проверяется наличие общих законов. Если такие законы найдены, между соответствующими вершинами (делами) в графе устанавливается ребро. Таким образом, чем больше законов пересекается между двумя делами, тем сильнее связь между ними, что может быть отражено в толщине ребра или других визуальных атрибутах графа.

4. После построения графа проводится его анализ, например, определяется количество вершин и рёбер. Это помогает оценить общий объем данных и степень взаимосвязанности дел.

После всех преобразований получен вот такой граф:

Заключение

Использование графа для визуализации и анализа взаимосвязей между судебными делами, основанных на общности применяемых законов, представляет собой мощный аналитический инструмент, который может значительно улучшить понимание сложных юридических процессов. Этот подход позволяет не только наглядно демонстрировать связи между различными делами, но и выявлять ключевые законы и статьи, которые часто фигурируют в разных контекстах.

Применение графовых моделей обеспечивает следующие преимущества:

1. Глубокий анализ взаимосвязей. Граф помогает увидеть, как связаны дела, что может указывать на общие юридические принципы или часто используемые законодательные нормы. Это особенно ценно для юристов и исследователей, которые стремятся понять закономерности в применении законов.

2. Выявление важных узлов. Анализ графа позволяет идентифицировать ключевые дела или законы, которые играют центральную роль в судебной системе. Такие узлы могут служить отправной точкой для дальнейших правовых исследований или для пересмотра и уточнения законодательства.

3. Улучшение законодательной работы. Понимание того, как законы пересекаются и взаимодействуют в различных делах, может помочь законодателям устранить противоречия в законодательстве и сделать правовую систему более эффективной и справедливой.

4. Образовательные возможности. Графы могут быть использованы в образовательных целях для демонстрации студентам-юристам сложности и многогранности судебной практики, а также для обучения их навыкам анализа юридических данных.

5. Оптимизация юридической практики. Для практикующих юристов анализ такого графа может помочь определить наиболее эффективные стратегии защиты или обвинения, исходя из ранее использованных успешных или неудачных юридических аргументаций.

В заключение, использование графа для анализа юридических взаимосвязей не только расширяет понимание текущего применения законодательства, но и открывает новые направления для улучшения и развития юридической системы в целом. Это делает его неотъемлемым инструментом в арсенале современного юриста или законодателя.

Список использованных источников

1. Собрание всех юридических дел по городу Москве – <https://mos-gorsud.ru/>
2. Обучающий видеохостинг - <https://www.youtube.com/>
3. Сервис IT-специалистов - Stack Overflow
4. Документация по регулярным выражениям на языке Python - <https://docs.python.org/3/library/re.html>
5. Документация по работе с playwright - <https://playwright.dev/python/docs/library>

Полный код проекта

https://github.com/BorisTerminator/Kursova\_2

https://github.com/BorisTerminator/Kursova\_2