Федеральное государственное образовательное бюджетное учреждение высшего образования
«Финансовый университет при Правительстве Российской Федерации»
Ярославский филиал

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ И ОБРАБОТКИ ДАННЫХ

Учебное пособие

Ярославль - 2024

УДК 004.41 (075.8)
ББК 32.973я73
Т 47

Печатается по плану изданий Ярославского филиала Финуниверситета

Рецензент:

Зайцев С.А., кандидат технических наук, доцент кафедры «Радиолокационных станций и автоматизированных систем управления» Военного учебного центра при Федеральном государственном автономном образовательном учреждении высшего образования «Национальный исследовательский университет «МИЭТ».

Т 47 Тихонов Д.В. Современные технологии прикладного программирования и обработки данных: учебное пособие / Д.В. Тихонов, М.О. Ермоленко. Ярославль: ООО «ПКФ «СОЮЗ-ПРЕСС», 2024. 232 с.

ISBN 978-5-6048031-9-6

УДК 004.41 (075.8)
ББК 32.973 я73

Настоящее учебное пособие разработано для учебно-методического обеспечения учебной дисциплины «Современные технологии прикладного программирования и обработки данных».

В учебном пособии раскрываются основные возможности языка программирования Python. Рассмотрены современные технологии для решения задач анализа данных, в том числе с применением машинного обучения и искусственного интеллекта. Представлен обзор технологий работы со структурированными данными, а также различные классы информационно-аналитических систем.

Пособие предназначено для использования в учебном процессе обучающимися по направлению подготовки 38.03.01 «Экономика».

ISBN 978-5-6048031-9-6

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире понятие «данные» являются неотъемлемой частью нашей повседневной жизни. От социальных сетей до космических исследований, от малого бизнеса до глобальных корпораций - везде приходится сталкиваться с необходимостью сбора, анализа и использования данных. Сфера анализа и обработки данных находится в постоянном развитии, открывая новые горизонты для исследований и инноваций.

Учебное пособие предназначено для студентов, специалистов и всех, кто интересуется современными технологиями в области обработки данных и программирования. В пособии представлен всесторонний обзор ключевых понятий и инструментов, на которых основано прикладное программирование, а также рассмотрен ряд наиболее востребованных инструментов на языке программирования Python. Проанализированы подходы, методы и приемы анализа данных, включая машинное обучение и искусственный интеллект.

Целью учебного пособия является не только предоставление теоретических знаний, но и обучение практическим навыкам, необходимым для успешного решения реальных задач в области анализа и обработки данных. В учебном пособии рассматриваются как классические методы, так и современные технологии, что позволит читателям получить необходимые знания и навыки для успешного освоения современных информационных систем.

Первая глава посвящена общим понятиям анализа данных. Представлены дистрибутив Anaconda, интерактивная оболочка Jupyter-notebook, Spyder, PyCharm и RStudio. Рассматриваются основные характеристики и возможности языка Python.

Во **второй главе** разобраны числа и арифметические операторы Python, а также основные синтаксические конструкции необходимые для работы со строками, циклами, списками, словарями.

Третья глава знакомит читателей с возможностями библиотек Pandas NumPy, SymPy, Matplotlib, TensorFlow, а также порядком решения базовых задач для решения задач анализа данных и машинного обучения.

Четвертая глава включает в себя обзор технологий работы со структурированными данными. Рассмотрены характеристики файлов и файловых систем, реляционных систем управления базами данных на примере языка SQL. Раскрыто понятие OLAP технологии. Описывается понятие хранилище данных (Data Warehouse).

В **пятой главе** разобраны технологии обработки больших данных (Big Data), а также особенности построения информационно-аналитических систем с применением алгоритмов машинного обучения.

Чтобы изложенный в учебном пособии материал лучше усваивался, теоретические сведения сопровождаются богатым иллюстративным материалом - таблицами и рисунками. В конце каждой главы приведён список контрольных вопросов.

Глава 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON

1.1 Общие понятия анализа данных

В информатике ключевыми являются такие понятия, как данные, информация и знания, которые, хоть и кажутся похожими, имеют существенные отличия.

Термин **«данные»**, произошедшие от латинского слова *data*, означает зафиксированные факты о событиях или явлениях, сохраняемые на различных носителях.

Термин **«информация»** происходит от слова *informatio* и обозначает процесс разъяснения или предоставления сведений, которые являются результатом обработки данных и приобретая значимость и смысл. В отличие от просто собранных данных, информация появляется при их анализе при решении конкретных задач, например, когда система управления базами данных выдаёт ответ на определённый запрос пользователя.

При рассмотрении понятия «**информация»** в контексте информационных систем можно выделить два аспекта: инфологический и датологический. **Инфологический аспект** касается смыслового содержания данных, абстрагируясь от способов их хранения, и направлен на определение необходимой информации о реальном мире и её характеристиках для системы.

Датологический аспект, в свою очередь, фокусируется на представлении данных в памяти системы, выборе подходящих форм их хранения и обработки. Это включает в себя разработку моделей данных, методов их преобразования и правил интерпретации.

Понятие **«знания»**, по сравнению с **«данными»** и **«информацией»**, представляет собой обработанную и проверенную информацию, которая уже была или может быть использована для принятия решений. Знания формируют опыт и компетенции, становясь со временем интеллектуальным капиталом. Знания могут быть представлены в виде документации, стандартов или различных учебных материалов, а также могут быть в неформальной форме, представляя собой накопленный опыт специалистов в определённой области. Таким образом понятия данные, информация и знания обеспечивают протекание процессов внутри информационных систем и играют ключевую роль в процессе принятия решений на их основе.

Решая задачи различных классов на электронных вычислительных машинах (ЭВМ) пользователи сталкиваются с большим разнообразием данных, которые необходимо представить в форме, наиболее соответствующей понятиям и объектам предметной области. Такие способы организации данных обычно не предусмотрены в языках программирования, поэтому их необходимо самостоятельно разрабатывать. Для этого в процессе разработки определяется структура и размер значения, его представление в оперативной памяти и множество допустимых значений, а также удобный набор операций для обработки этих данных, которые можно описать как набор подпрограмм. Способы организации данных, ориентированные на представление объектов и понятий предметной области, называются абстрактными типами данных или пользовательскими.

В современном мире роль данных сложно преувеличить. Данные необходимы для выполнения научных исследований, их используют при решении аналитических и других задач в различных областях. Данные необходимы для принятия некоторых частных решений, но, когда необходимо проанализировать большие объёмы информации или сделать сложный прогноз, необходимо применять набор данных, который может значительно повысить эффективность и точность выполнения процессов.

Набор данных, или **Dataset** это совокупность данных, полученных из определенного источника данных, которые были обработаны и структурированы в соответствии с конкретными задачами аналитики и визуализации. В наборе данных содержится информация об источнике данных, а также вид проведённого преобразования информации.

Структура данных - это организованная информация, которую можно описать, создать и обработать с помощью программ. Вместе со структурами управления вычислениями она является фундаментальной составляющей программирования. Данные делятся на два класса: со статической и динамической структурой.

Данные со статической структурой - это данные, в которых расположение и взаимосвязь элементов всегда постоянное. **Данные с динамической структурой** формируют своё внутреннее строение по определённому закону, при этом количество элементов, их расположение и взаимосвязи могут изменяться во время выполнения программы, но перед использованием должны быть инициализированы.

Наборы данных можно классифицировать различными способами. Среди способов классификации - **деление по типу данных**: числовые (содержат числа и используются для количественного анализа), текстовые

(посты, текстовые сообщения и документы), мультимедийные (изображения, видео и аудиофайлы) и данные временных рядов (собираются за определённый период времени для анализа тенденций и закономерностей). К этому же типу относятся и пространственные данные, то есть включающие информацию с географической привязкой, например, данные GPS.

Ещё один способ классификации - **разделение по структуре данных**. Существуют структурированные наборы (организованы в определённые структуры, чтобы упростить запрос и анализ данных), неструктурированные (не имеют строгой схемы) и гибридные наборы данных, которые содержат как структурированные, так и неструктурированные данные.

Третий способ классификации связан со **статистикой**. В зависимости от количества переменных выделяют двумерные (включают две переменные данных) и многомерные наборы данных (три или более переменных). Среди них выделяют числовые, состоящие лишь из числовых значений, и категориальные, состоящие из категориальных переменных, которые могут принимать лишь ограниченный набор значений. Кроме того, среди статистических наборов данных выделяют корреляционные, включающие переменные, связанные друг с другом.

Также существуют наборы данных для обучения модели машинного обучения, которые используются, чтобы обучить модель. Наборы данных для валидации, применяемые, чтобы снизить переобучение и повысить точность модели. Наборы для тестирования, способные подтвердить точность конечного результата модели.

1.2 Основные характеристики языка программирования Python

История развития языка программирования Python насчитывает немногим более двадцати лет, а его автор, *Гвидо Ван Россум*, вероятно, не предполагал, что разработанный им простой интерпретирующий язык получит такое широкое распространение. Язык быстро развивался, но при этом сохранял совместимость с ранее написанными программами. В течение долгого времени рабочими были версии *2.x.* Конечно, в язык вносились небольшие улучшения, а более серьёзные изменения осуществлялись через модуль **__future**, который позволял программистам разрабатывать программы с учётом будущих изменений в языке.

Однако, вместе с переходом к версиям *3.x* были введены более принципиальные изменения, нарушая совместимость версий *2.x*. Для

облегчения перехода на новые версии языка существуют специальные программы, автоматизирующие большую часть работы программиста.

Сейчас разработчики языка программирования Python взяли паузу. Тем временем были созданы различные среды для разработки и выполнения программ на языке Python. Самая простая среда – IDLE (*Integrated Developvent and Learning Enviroment*) представляет собой интегрированную среду разработки и обучения на языке Python. Она поставляется вместе с интерпретатором языка и реализована практически для всех операционных систем. Python - это интерпретируемый язык программирования сверхвысокого уровня. Основные особенности языка Python:

- в Python отсутствуют описания переменных: тип переменной определяется в момент её инициализации конкретным значением; при присваивании переменным нового значения их тип может измениться;
- в Python отсутствуют указатели: все динамические объекты реализованы через ссылки;
- Python имеет объектно-ориентированный характер: переменные являются объектами некоторого типа, а для работы с переменными используются методы соответствующего типа;
- язык Python поддерживает различные парадигмы программирования: процедурное, объектно-ориентированное и функциональное;

Python и среда разработки IDLE являются свободным программным обеспечением и распространяются на условиях GPL (*General Public License*) - лицензии.

Установка Python. Чтобы создавать программы на Python, необходим интерпретатор. Его можно скачать на официальной странице *https://www.python.org/downloads/*. На странице необходимо выбрать ссылку на загрузку последней версии языка Python (рисунок 1).

При нажатии на кнопку «Скачать Python 3.12.3» будет загружен установщик Python для соответствующей операционной системы (ОС). Необходимо отметить, что Windows 7 и более ранние версии операционных систем не поддерживаются.

При запуске инсталлятора на ОС Windows открывается окно мастера установки (рисунок 2). В нем можно указать путь, по которому будет происходить установка интерпретатора. Также необходимо отметить флажок «Add Python.exe to PATH» в самом низу, чтобы добавить путь к интерпретатору в переменные среды. После того как интерпретатор Python будет установлен, можно приступать к работе на Python (рисунок 3).

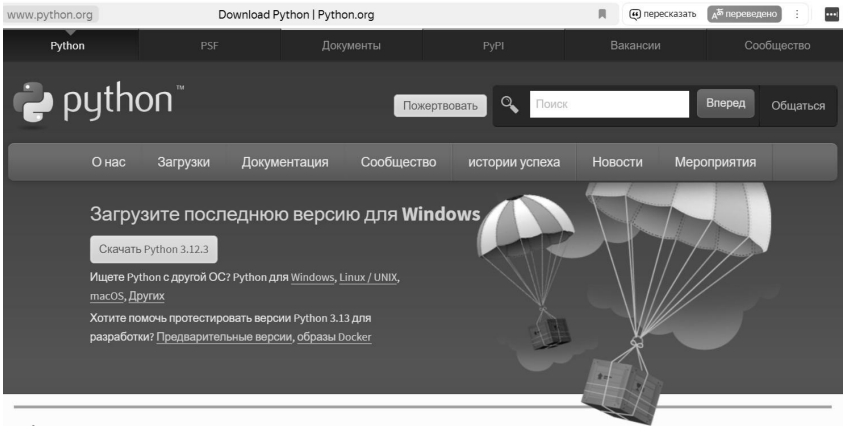

Рисунок 1. Страница установки Python

Интерпретатор Python обладает обширным набором функций, позволяющим создавать приложения на этом языке программирования. Однако в некоторых случаях даже этого функционала может быть недостаточно для решения определённых задач. К счастью, благодаря большому сообществу разработчиков, использующих Python по всему миру, существует развитая экосистема пакетов и библиотек, которые можно применять для различных целей. Это делает Python ещё более мощным и универсальным инструментом для разработки.

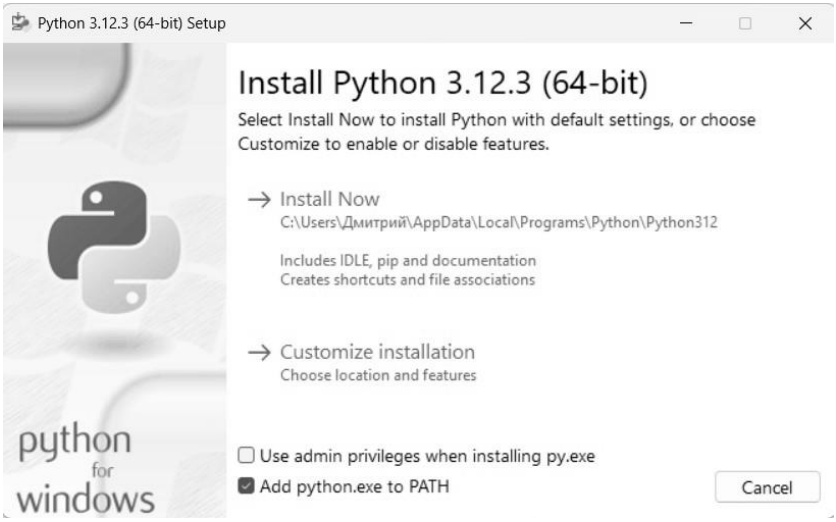

Рисунок 2. Окно мастера установки Python

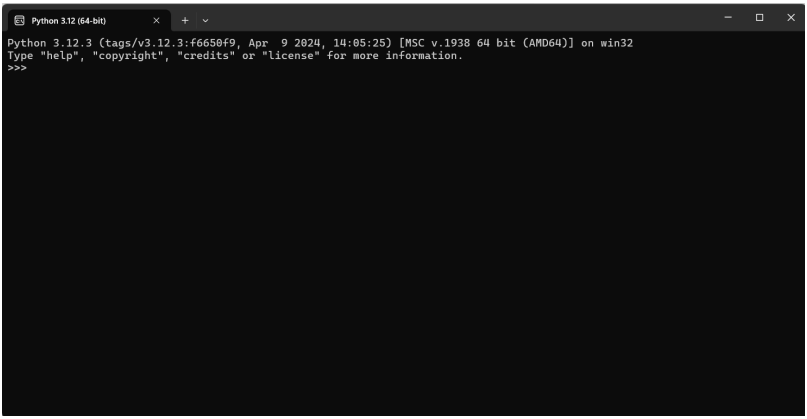

Рисунок 3. Интерпретатор Python

В этом разделе будут рассмотрены библиотеки Python. В зависимости от их назначения библиотеки условно можно разделить на несколько основных групп.

К первой группе относятся библиотеки, предназначенные для веб-разработок. Они представляют собой готовые функции и модули для облегчённого и эффективного создания веб-приложений. К ним относятся:

1. **Библиотека HTTPX**. Эта популярная библиотека, которая позволяет создавать сайты или веб-приложения. В отличие от Requests, HTTPX даёт возможность работать не только с синхронными, но и с асинхронными HTTP-запросами, что делает её более гибкой и мощной.

Ещё одно преимущество HTTPX заключается в том, что она поддерживает современную версию протокола – HTTP/3, который обеспечивает более быстрый и эффективный обмен данными. Кроме того, HTTPX проста в использовании, что особенно ценно для начинающих пользователей.

Для того, чтобы установить эту библиотеку, достаточно выполнить команду `pip install httpx`.

2. **Библиотека Celery**. Это инструмент, который используется для работы с фоновыми задачами. Она позволяет выстраивать их в очередь и распределять выполнение между разными процессорами и устройствами, что помогает уменьшить нагрузку на процессор и выполнять трудоемкие задачи без ущерба производительности.

Для установки библиотеки необходимо выполнить команду `pip install celery`.

3. **Библиотека Scrapy**. Библиотека предназначена для сбора данных и используется при разработке приложений. С её помощью можно создавать поисковых роботов, также известных как веб-краулеры, и другие алгоритмы для сбора информации. Scrapy поддерживает асинхронность, что позволяет эффективно решать разнообразные задачи - от загрузки веб-страниц до их обработки и сохранения в разных форматах.

Для установки Scrapy используется команда `pip install scrapy`.

4. **Библиотека Dash**. Это библиотека Python, предназначенная для работы над веб-приложениями, которая предлагает доступный инструментарий для создания графических интерфейсов (GUI), благодаря чему пользователь может легко взаимодействовать с данными.

Для установки Dash используется команда `pip install dash`.

Ко второй группе относятся библиотеки, предназначенные для решеня задач машинного обучения. Такие библиотеки, предназначены для обучения различных моделей данных для того, чтобы затем использовать их для классификации, регрессии, кластеризации и других задач.

1. **Библиотека NumPy**. Библиотека, которая позволяет хранить и изменять данные в *n-мерных массивах*, что в некоторых ситуациях может быть более надёжным решением, чем использование обычных списков в Python. Основная функция NumPy - способность выполнять сложные математические преобразования.

Для установки NumPy используется команда `pip install numpy`.

2. **Библиотека Pandas.** Основная библиотека Python, предназначенная для решения сложных задач машинного обучения, связанных с подготовкой и первичным анализом данных. Она позволяет удобно фильтровать, объединять и группировать данные, что делает её незаменимым инструментом для специалистов по машинному обучению. С помощью Pandas можно строить модели интерпретации данных любого уровня сложности.

Чтобы установить эту библиотеку, необходимо выполнить команду в терминале:

`pip install pandas`.

3. **Библиотека TensorFlow**. Библиотека, предназначенная для разработки нейронных сетей. В ней используются тензоры - многомерные массивы, которые дают возможность работать с несколькими сетями одновременно. Кроме того, TensorFlow часто применяется для распознавания изображений и рукописного текста.

Для установки библиотеки используется команда: `pip install tensorflow`.

4. **Библиотека LightGBM**. Библиотека для машинного обучения, разработанная специалистами Microsoft. Её главная особенность заключается в том, что она позволяет быстро реализовать градиентный бустинг - принцип машинного обучения, который помогает создавать новые алгоритмы с использованием многоуровневых решений.

Чтобы установить LightGBM, нужно выполнить команду: `pip install lightgbm`.

В условиях постоянно увеличивающегося объёма данных их обработка без применения специальных инструментов может стать трудоёмким и долгим процессом. В то же время в Python существует множество библиотек, решающих подобные задачи. Наличие таких инструментов значительно упрощает работу аналитиков и исследователей, позволяя эффективно работать с данными и получать необходимые результаты.

Среди аналитиков и научных работников востребованы библиотеки для форматирования и очистки данных на Python, которые помогают выполнять обработку данных, приводить их к нужному виду, удалять пропуски и выбросы, объединять разные источники данных и т.д. Эти библиотеки относятся к ***третьей группе***.

1. **Библиотека Dora**. Библиотека, предназначенная для упрощения процесса анализа данных, что делает её важным инструментом для специалистов в области Data Science. С её помощью можно легко преобразовывать различные данные, изменять и удалять столбцы, выделять, извлекать и визуализировать признаки, а также выполнять множество других задач, связанных с обработкой данных. Для установки Dora необходимо выполнить команду `pip install dora`.

2. **Библиотека** Datacleaner предназначена для автоматической очистки и подготовки данных к анализу. Она позволяет выполнять различные операции с данными, такие как удаление строк без указанного значения, кодирование нечисловых переменных и другие. Datacleaner может работать с фреймами Pandas и имеет простой интерфейс, что делает её удобной для использования как опытными, так и начинающими пользователями.

Для того чтобы установить Datacleaner, необходимо выполнить команду `pip install datacleaner`.

3. **Библиотека Tabulate**. Это инструмент, который позволяет создавать таблицы с удобным дизайном. Он предоставляет множество функций форматирования, которые помогают эффективно представлять словари, списки и двумерные массивы из библиотек NumPy и Pandas.

Возможности Tabulate не ограничиваются только консолью: данные также можно экспортировать в различных веб-форматах, таких как HTML и Markdown Extra.

Чтобы использовать Tabulate, необходимо установить его через pip: `pip install tabulate`.

4. **Библиотека Scrubadub**. Это библиотека, которая особенно полезна при работе с конфиденциальными данными. Она предоставляет множество гибких инструментов для удаления из массива данных имён, телефонных номеров, URL-адресов, идентификаторов и других важных сведений. Чтобы использовать эту библиотеку, необходимо выполнить команду в терминале: `pip install scrubadub`.

Четвертая группа библиотек Python представляют собой библиотеки для визуализации данных. Они позволяют создавать привлекательные и понятные изображения, наглядно представляющие информацию. С их помощью можно сделать данные более воспринимаемыми, превратить их в разнообразные графики, диаграммы, дашборды и другие визуальные элементы.

1. **Библиотека Matplotlib**. Одна из ключевых библиотек для визуализации данных на языке Python, которая служит основой для множества других инструментов, таких как Cartopy и Seaborn. Отличительной чертой Matplotlib является его понятный объектно-ориентированный интерфейс и удобный Application Programming Interface (API) для встраивания графиков в разнообразные приложения. Для установки Matplotlib в среде Python используется команда `pip install matplotlib`.

2. **Библиотека Altair**. Библиотека на Python, разработанная для статистической визуализации данных. С Altair можно работать с трёхмерной графикой, но библиотека станет отличным инструментом для создания сложных двумерных графиков.

Чтобы установить Altair на свой компьютер, можно использовать команду `pip install altair`.

3. **Библиотека Bokeh**. Библиотека для визуализации данных, оптимизированная для работы в браузерах, которая предназначена для поддержки веб-приложений. Она работает с объектами формата JSON и оптимизирована для обработки данных в реальном времени. Bokeh часто используют для создания различных интерактивных визуализаций.

Установить данную библиотеку можно при помощи команды: `pip install bokeh`.

4. **Библиотека Leather**. Новая библиотека для визуализации данных, написанная на Python. Она особенно полезна, когда нужно быстро решить задачу визуализации данных. В отличие от других подобных библиотек, Leather может работать с разнообразными данными и создавать векторную графику для их представления.

Чтобы использовать библиотеку, достаточно выполнить команду в терминале: `pip install leather`.

1.3 Дистрибутив Anaconda

Anaconda Python - это дистрибутив, содержащий языки программирования Python и R с открытым исходным кодом. Он включает в себя набор свободных библиотек, систему управления пакетами и другие компоненты и предназначен для использования на ОС Windows, Linux и MacOS (рисунок 4).

Рисунок 4. Логотип Anaconda Python

Дистрибутив применяют для проведения научных и инженерных расчётов, обработки данных, прогнозной аналитики и машинного обучения. Разработкой и поддержкой Anaconda Python занимается компания Anaconda Inc. Она была основана *Питером Вангом* и *Трэвисом Олифантом* в 2012 году. Дистрибутив разработан таким образом, чтобы упростить установку пакетов и управление ими.

Программа Anaconda представляет собой комплексный набор инструментов для работы с библиотеками Python, который включает программу установки и несколько ключевых компонентов:

1. **Библиотеки Python** - каждая библиотека содержит группу функций и инструментов, предназначенных для решения определённых задач. Например, работы с большими объёмами данных, астрономических вычислений, обработки изображений, создания и обучения нейронных сетей, выполнения инженерных расчётов, статистического анализа и многого другого. В Anaconda представлено около 1500 библиотек различной тематики - от научных исследований до инженерных приложений.

2. **Менеджер пакетов Conda** - это программное обеспечение (ПО), разработанное для установки, обновления и управления пакетами программного обеспечения и библиотеками. Менеджер пакетов Conda написан на языке Python, но может работать с проектами, созданными на любом языке программирования. Уникально то, что с помощью менеджера Conda также можно установить Python. Из-за своей универсальности и удобства использования Conda была выделена в отдельный дистрибутив с открытым исходным кодом.

3. **Менеджер виртуальной среды** - программа, которая позволяет создавать, изменять, удалять и отслеживать виртуальные машины. Управление этими изолированными средами осуществляется через единую консоль. Виртуализация полезна при работе над несколькими проектами одновременно, поскольку предотвращает их взаимодействие друг с другом и обеспечивает высокую точность результатов.

4. **GUI Anaconda Navigator** - представляет собой графический интерфейс пользователя, который упрощает работу с библиотеками, делая взаимодействие с ними более наглядным и интуитивно понятным. Вместо текстовых команд пользователь может использовать визуальные элементы, такие как блоки, модули и графики.

В дистрибутив Anaconda входит несколько полезных программ. К ним относятся: Jupyter Notebook и JupyterLab позволяют исполнять код на Python и на других языках, а также обрабатывать данные. Spyder и PyCharm представляют собой интегрированную среду разработки (IDE), которая объединяет в себе редактор кода наподобие программы Atom или Sublime Text с дополнительными функциями.

Структуру дистрибутива Anaconda можно представить следующим образом (рисунок 5):

Рисунок 5. Структуру дистрибутива Anaconda

К ним относятся возможности автодополнения, компиляции и интерпретации, анализа ошибок, отладки, подключения к базам данных и другие. RStudio - это интегрированная среда разработки для программирования на языке R.

Для установки дистрибутива Anaconda его необходимо загрузить с официального сайта *https://www.anaconda.com/distribution/*. Затем запустить установщик (рисунок 6).

Рисунок 6. Запуск установки Anaconda

На одном из этапов установки предлагается установить четыре флажка (рисунок 7):

- ☐ Создать ярлык на рабочем столе;
- ☐ Добавить Anaconda в системную переменную PATH;
- ☐ Сделать дистрибутив Anaconda версией, которая будет обнаруживаться Windows по умолчанию;
- ☐ Очистить кэш для экономии места на диске.

Рекомендуется отмечать первый и последний пункт (создать ярлык на рабочем столе и очистить кэш для экономии места на диске). Далее необходимо дождаться завершения установки (рисунок 8).

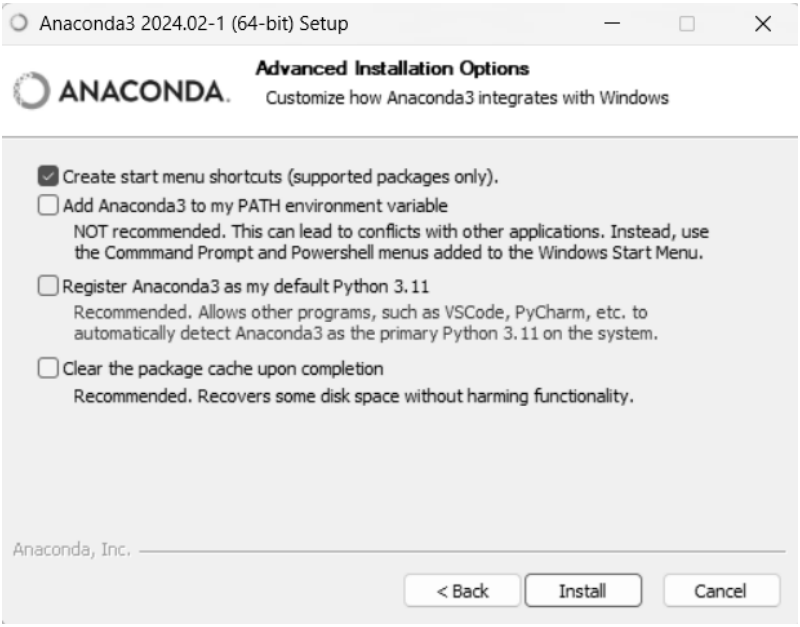

Рисунок 7. Этап установки дистрибутива Anaconda

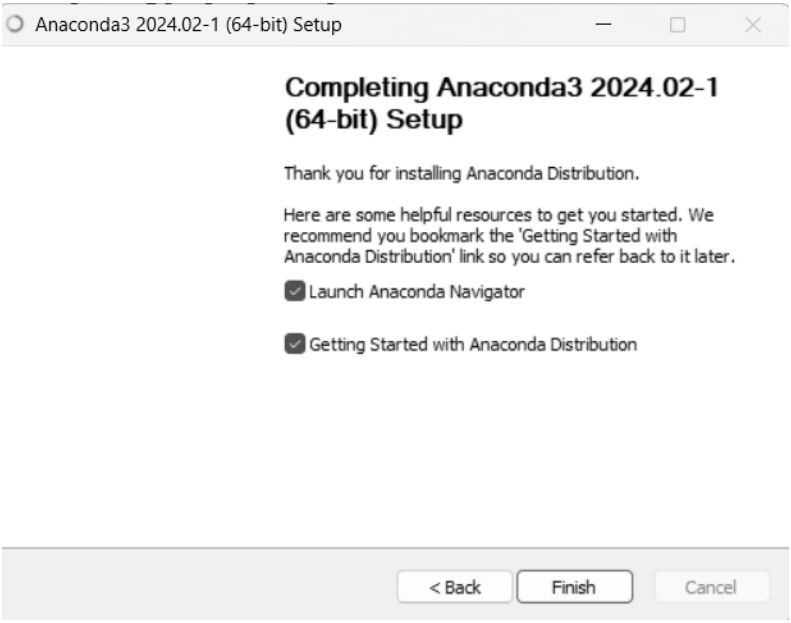

Рисунок 8. Завершение установки дистрибутива Anaconda

Среди преимуществ дистрибутива Anaconda является то, что он распространяется как свободное программное обеспечение и имеет открытый исходный код. Это предоставляет возможность пользователям просматривать и изменять его в соответствии со своими потребностями. В состав Anaconda входит большое количество предустановленных библиотек – их более полутора тысяч. Это упрощает работу пользователю, так как нет необходимости искать и устанавливать определённые библиотеки отдельно.

Кроме того, одним из ее ключевых преимуществ Anaconda является доступность документации. Так как это продукт с открытым исходным кодом, в нем есть множество руководств, инструкций по установке, настройке и использованию от официальных разработчиков. Этот факт облегчает изучение и освоение Anaconda всем желающим.

Ещё одна особенность Anaconda - её универсальность. Она обладает возможностью установки на различные ОС, такие, как Windows, Linux и MacOS. Наконец, Anaconda предлагает два варианта установления и управления программными пакетами, что делает эту систему привлекательной для пользователей с разным уровнем опыта. Те пользователи, которые только знакомятся с Python, могут использовать графический интерфейс для управления компонентами и библиотеками. Более опытным пользователям будет удобнее работать с помощью командной консоли Conda. В любом случае каждый пользователь может выбрать оптимальный для него вариант взаимодействия с системой.

1.4 Интерактивная оболочка Jupyter Notebook

Jupyter Notebook - это веб-оболочка для Ipython (ранее назывался IPython Notebook), представляющая собой веб-приложение с открытым исходным кодом, которое позволяет создавать и совместно использовать документы, включающие в себя функционирующий код, формулы, визуализацию данных и форматированный текст.

Изначально IPython Notebook поддерживал только язык программирования Python. Благодаря оболочки Jupyter Notebook стало возможным использование таких языков программирования, как Python, R, Julia, Scala. В сфере Data Science уже на протяжении нескольких лет одним из востребованных инструментов для анализа данных и быстрого создания прототипов считается Jupyter Notebook. Jupyter Notebook состоит из двух основных частей: веб-приложения и файлов-блокнотов, предназначенных

для работы с исходным кодом программы, его запуска и вывода данных в различных форматах.

Для экспорта блокнотов доступны два формата: PDF и HTML. В веб-приложении можно выполнять следующие действия:

- запускать и редактировать код непосредственно в браузере;
- показывать результаты вычислений с помощью разных схем и графиков;
- использовать язык разметки Markdown и LaTeX.

Преимущество этого инструмента заключается в том, что можно разделить код на отдельные части и работать над ними в любом порядке. Например, можно написать часть программного кода и сразу же проверить, как он работает. При этом нет необходимости запускать остальные фрагменты кода - результат отображается сразу под соответствующей частью кода. Так специалисты по Data Science получают предварительные результаты исследований, строят графики и диаграммы. Принцип работы веб-оболочки представлен на рисунке 9.

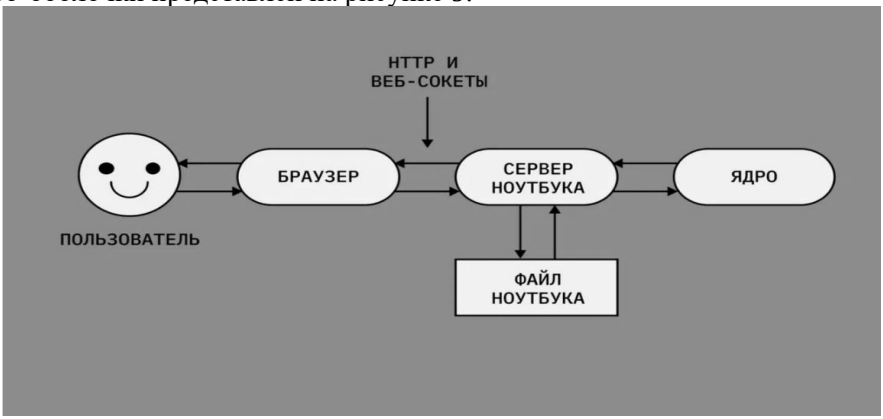

Рисунок 9. Принцип работы веб-оболочки Jupyter Notebook

1.5 Spyder и PyCharm

Spyder - это среда разработки на Python, предназначенная для научного программирования. Она способна работать с несколькими операционными системами и имеет открытый исходный код. Spyder интегрируется с рядом пакетов из Python Scientific Stack, включая NumPy, SciPy, Matplotlib, Pandas, IPython, SymPy и Cython. Spyder расширяется с помощью подключаемых модулей, включает поддержку интерактивных инструментов для проверки

данных и инструменты для обеспечения качества и анализа, такие как Pyflakes, Pylint и Rope.

Интегрированная среда разработки доступна на всех платформах: она использует инструментарий Qt для графического интерфейса и поэтому может быть использована в Windows, MacOS (через MacPorts) и Linux. Также Spyder входит в состав пакета программ Anaconda в качестве основной интегрированную среду разработки (IDE).

Spyder IDE представляет собой редактор исходного кода, обладающий рядом особенностей:

- подсветка синтаксиса, анализ и автозавершение кода;
- возможность проверки и редактирования переменных из графического интерфейса;
- поддержка нескольких консолей IPython;
- окно справки, которое автоматически или по мере необходимости извлекает документацию в формате RTF по функциям, классам и методам;
- пошаговый отладчик, подключенный к IPdb;
- профилировщик времени выполнения для тестирования кода;
- статический анализ кода, предоставляемый Pylint;
- сопровождение проекта, позволяющее одновременно работать над несколькими направлениями развития;
- интегрированный файловый менеджер для взаимодействия с файлами проекта;
- функция поиска в файлах, выполняющая поиск регулярных выражений в определённом диапазоне;
- браузер интерактивной справки, позволяющий пользователям просматривать документацию по пакетам;
- журнал истории, записывающий каждую пользовательскую команду, введённую на каждой консоли;
- внутренняя консоль, предоставляющая самопроверку и контроль над собственной работой Spyder.

Возможности среды разработки Python для Windows можно значительно расширить, благодаря плагинам и API. Также среду можно использовать в качестве библиотеки расширений PyQt5, позволяющей применять функциональные возможности этого инструмента и встраивать его компоненты, такие как интерактивная консоль, в собственное программное обеспечение.

Spyder IDE включает в себя следующие основные компоненты:

1. Редактор исходного кода - это эффективный редактор с многоязычным интерфейсом и браузером функций/классов. Код можно

анализировать в реальном времени с помощью инструментов pyflakes, pylint и pycodestyle; автоматически завершать код с помощью jedi и rope; разделять горизонтально и вертикально и переходить к определению.

2. Интерактивная консоль - можно запустить в ней любой код по строкам, ячейкам или файлам и визуализировать графики прямо в выводе или в интерактивных окнах. Консоль IPython поддерживает любое количество рабочих областей и отладку. Всё это можно делать в графическом интерфейсе.

3. Средство просмотра документации - с помощью Sphinx можно визуализировать документацию для любого класса или функции, будь то созданные пользователем или внешние. Это можно сделать как из консоли, так и из редактора.

4. Проводник переменных - позволяет проверять любые переменные, объекты или функции, созданные в ходе сеанса. Поддерживает множество типов данных, включая числовые, строковые, булевы, списки, кортежи, словари Python, временные интервалы, массивы Numpy, индекс, серии, кадры данных Pandas, изображения PIL и другие. В проводнике возможно редактирование и взаимодействие с объектами.

5. Инструменты разработки – в Spyder IDE с помощью статического анализатора можно изучить код, отследить его выполнение через интерактивный отладчик и оценить его производительность с помощью профилировщика. Упорядочить работу помогут поддержка проектов, встроенный файловый проводник и функция поиска в файлах. Поиск возможен с применением регулярных выражений и работает во всех проектах.

На сайте релизов Spyder IDE можно скачать облегчённые версии программы. Они не включают в себя некоторые необязательные, но рекомендуемые зависимости, такие как NumPy, SciPy и Pandas. Это значит, что часть функций Variable Explorer, таких как мастер импорта графических данных и поддержка расширенного отображения и редактирования массивов NumPy и Pandas DataFrames, будет недоступна. Авторы Spyder рекомендуют пользоваться полной версией программы, если только максимальный размер загрузки и использования оперативной памяти не являются принципиальными.

Для запуска Spyder нужны две основные зависимости:

1. Python 3.6+ основной язык, на котором написан Spyder;
2. PyQt5 5.6+ привязки Python для Qt, которые используются для графического интерфейса Spyder.

PyCharm представляет собой интегрированную среду разработки для языка программирования Python. PyCharm является мощным и многофункциональным инструментом. PyCharm -интегрированная среда разработки для Python -существует для нескольких операционных систем: Windows, Linux и macOS. Эта среда поддерживает различные версии Python: и 2.x, и 3.x. Благодаря широким возможностям PyCharm, разработка на Python становится быстрее и эффективнее.

Рассмотрим некоторые из возможностей PyCharm:

Создание проекта: в PyCharm можно создавать полноценные проекты, при этом среда выделяет под проект отдельную папку, где хранятся все связанные файлы и компоненты. Структура проекта отображается в левой части интерфейса, позволяя легко переключаться между файлами в проекте.

Написание кода: внутри проекта можно создавать файлы с расширением, соответствующим выбранному языку программирования, и писать в них код. Синтаксис подсвечивается автоматически, параметры подсветки настраиваются. PyCharm дополнительно проверяет написанный код на ошибки. Также обеспечивается поддержка чистого написания кода.

Запуск: среда выполняет запуск прямо внутри себя благодаря подключению IDE к интерпретатору или компилятору нужного языка. Это не требует открытия консоли или использования сторонних приложений. Для запуска достаточно нажать на соответствующую кнопку.

Отладка: PyCharam предоставляет инструменты для отладки кода. Можно настроить режим отладки так, что будут отображаться значения переменных в нужный момент времени. Можно также остановить выполнение кода на определённой строке и проверить работу кода в этом месте. Есть функция пошагового выполнения кода – программа выполняет одну строчку кода и останавливается, позволяя проверить корректность работы на этом участке.

Тестирование: Python широко используется для автоматического тестирования. В PyCharm есть инструменты для тестирования, а также возможность подключения модулей для автоматизации тестирования.

Корректировка: чтобы обеспечить читаемость и понятность кода, необходимо следовать определённым структурным правилам. PyCharm следит за тем, чтобы код соответствовал этим правилам. Также он может автоматически расставлять переносы строк и отступы и дополнять написанное: необходимо вводить только часть команды, а PyCharm предложит возможные варианты завершения.

Установка библиотек и фреймворков: поддержка популярных фреймвормов уже предусмотрена в некоторых версиях PyCharm. Другие

версии предоставляют возможность быстрой загрузки и интеграции фреймворком с проектом. Это упрощает процесс установки и настройки окружения.

Помимо написания кода на Python, в данной среде разработки можно устанавливать модули и плагины, которые позволяют расширить функционал IDE. К примеру, модули помогают проверять читаемость кода, подсказывают с помощью искусственного интеллекта, расставляют недостающие скобки и многое другое. Некоторые плагины позволяют изменить интерфейс среды разработки. Другие – расширяют функционал среды. После установки модулей и плагинов ими можно пользоваться как частью IDE.

В PyCharm можно писать и на других языках, помимо Python. Так, для веб-разработки в среде разработки предусмотрен JavaScript. Также поддерживаются языки SQL для работы с базами данных, HTML и CSS для верстки сайтов, а также TypeScript и CoffeeScript (языки, основанные на JavaScript) и различные JS-фреймворки. Среда позволяет использовать шаблонизаторы для создания шаблонов веб-страниц. С ними PyCharm также умеет работать.

PyCharm имеет две основные версии: бесплатную Community и платную Pro. Версия Pro предназначена для профессиаоналов и предлагает более широкую функциональность. В версии Community можно учиться, писать код для себя или для небольших проектов, и для большинства задач её функционала будет достаточно.

Версия Community не поддерживает некоторые технологии, такие как JavaScript, CSS и другие веб-технологии, работа возможна только с «чистым» Python и его вариациями, а также с HTML, XML, JSON и некоторыми другими форматами. Также в ней есть поддержка только фреймворков для «питона». В то же время версия PyCharm Pro, помимо Python фреймворков, поддерживает ещё и React, Angular и другие инструменты веб-разработки, основанные на JavaScript.

Кроме того, в версии Community нет встроенных инструментов для работы с базами данных, тогда как в версии Pro эти инструменты доступны.

В версии PyCharm Community инструменты для развёртывания и контроля версий устанавливаются отдельно как плагины. В PyCharm Professional они предустановлены.

Возможности совместной разработки в бесплатной версии PyCham ограничены: не более трёх участников и сессии не длиннее 30 минут.

1.6 RStudio

Язык R - это мощный высокоуровневый объектно-ориентированный язык программирования и среда для статистических вычислений, визуализации исходных и расчётных данных. Он позволяет решить множество задач в области обработки данных и представляет собой бесплатную программу с открытым исходным кодом, которая работает под управлением наиболее популярных операционных систем, таких как Microsoft Windows, MacOS, Linux и Unix. Язык R поддерживает тысячи специализированных модулей и утилит.

Одна из самых важных особенностей языка R - эффективная реализация векторных операций, что позволяет использовать компактную запись при обработке больших объёмов данных. Благодаря этому R является эффективным инструментом для извлечения полезной информации, в том числе из больших данных (Big Data). Это делает его удобным и эффективным средством для обучения технологии анализа, обработки и визуализации данных.

Сегодня язык R - один из основных статистических инструментов в мире. Его активно используют в таких областях, как генетика, молекулярная биология, биоинформатика, науки об окружающей среде (например, экология и метеорология), а также экономические и сельскохозяйственные дисциплины. Кроме того, R набирает популярность в обработке медицинских данных, постепенно вытесняя коммерческие пакеты.

Существует несколько программ-оболочек для удобства работы с R, они могут отличаться по внешнему виду и функциональности. Самые популярные варианты - Rgui, RStudio и R, запущенный в терминале Linux/UNIX в командной строке.

RStudio - интегрированная среда разработки (*Integrated Development Environment, IDE*) для работы с языком программирования R. Это очень мощный инструмент, предоставляющий широкие возможности для создания, компиляции и тестирования проектов, работы с данными и их визуализации. Основные возможности RStudio:

Интерфейс и структурирование кода. В RStudio используется интуитивный и удобный интерфейс, который позволяет быстро и легко писать и редактировать код. Кроме того, RStudio предлагает различные способы структурирования кода, включая окружение кода блоками, использование комментариев, отступов и т.д.

Инструмент широко известен благодаря возможностям для работы с графиками: пользователи могут создавать профессиональные графики и

диаграммы в интерактивном режиме. Отчёты становятся более наглядными для понимания и дальнейшей работы.

В RStudio предусмотрено автозавершение кода, поиск и переход по функциям, что упрощает разработку и уменьшает количество ошибок. Также доступны автоформатирование кода по тем или иным стилям и подсветка синтаксиса. Для опытных пользователей существует возможность настраивать это под свои потребности.

Для выполнения анализа данных в RStudio есть инструменты статистической обработки и моделирования. Можно осуществлять поиск пакетов, импортировать и экспортировать данные, а также выполнять обмен данными между различными форматами. Ещё одна полезная возможность RStudio -управление проектами. В этом режиме можно следить за изменениями в своём продукте и управлять версиями, сохраняя свои наработки без опасения потерять данные.

Консоль RStudio (Console) предоставляет широкий спектр функций, которые делают работу с языком программирования R удобной и эффективной (рисунок 10).

Редактор кода RStudio предоставляет множество функций для эффективной работы, включая подсветку кода и другие специализированные опции для работы с различными типами файлов, такими как R-скрипты, документы Sweave, документы TeX.

RStudio также предлагает автоматическое завершение кода, одновременное редактирование нескольких файлов, поиск и замену определённых частей кода. Помимо этого, у пользователей есть возможность выполнять код прямо в окне редактора, что делает его популярным выбором для многих учащихся.

Когда выполняется код в RStudio, команды отправляются в Консоль, где отображается и результат их выполнения (смотрите рисунок 10).

Во время работы RStudio создаёт базу данных всех команд, введённых пользователем в Консоль. Пользователь может просматривать эти данные с помощью панели History (История). Если все файлы для определённого проекта хранятся в одной папке, рекомендуется сделать эту папку исходной для работы. RStudio автоматически определит рабочую папку на основе папки, где находится открываемый файл.

Благодаря функционалу RStudio пользователи могут организовать работу по проектам, каждый из которых будет иметь свою рабочую директорию, рабочее пространство, историю и скрипты. Проекты RStudio связаны с рабочими директориями R. Это даёт возможность создать проект:

- В новой директории.

• На основе существующей директории, где уже есть скрипты с R-кодом и данные.

• Путем копирования файлов из существующей онлайн-системы контроля версий.

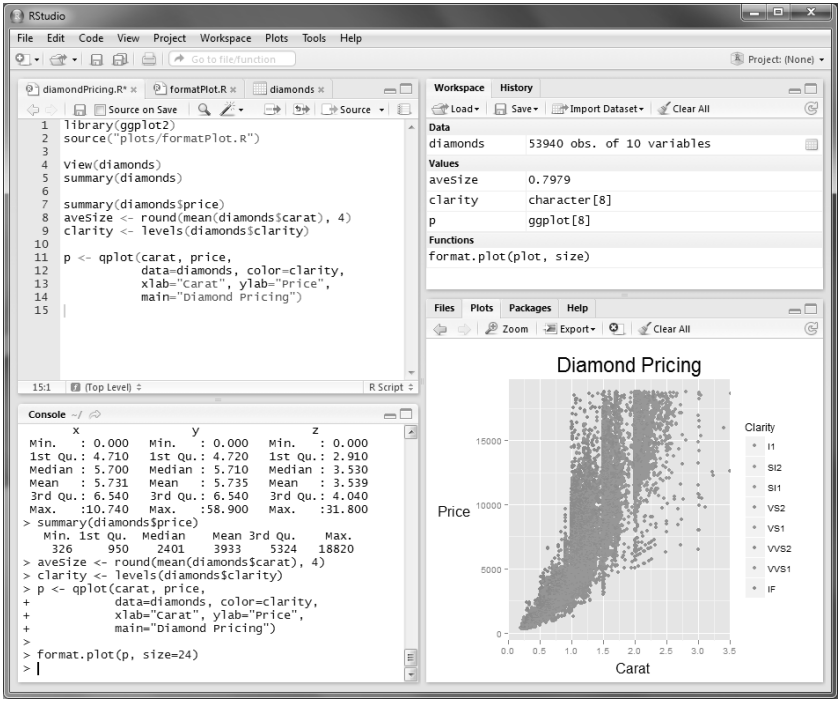

Рисунок 10. Консоль RStudio

Для создания нового проекта используйте команду New Project (Новый Проект), доступную на закладке Projects главного меню и на панели инструментов в правом нижнем углу окна программы. Также есть несколько опций для настройки проекта в RStudio. Эти параметры доступны через команду Project Options в разделе Project главного меню программы.

В общем, язык R, IDE RStudio и библиотека CRANE предоставляют возможности на уровне коммерческих продуктов и могут быть использованы для междисциплинарного взаимодействия, в научных исследованиях и при написании выпускных квалификационных работ студентами всех уровней подготовки.

Контрольные вопросы по главе 1

1. Дайте определения понятиям «данные», «информация», «знания»
2. Что такое набор данных?
3. Какими свойствами обладает набор данных? Охарактеризуйте каждое свойство.
4. Дайте классификацию набору данных.
5. Охарактеризуйте язык программирования Python. Какими основные свойства языка вы знаете?
6. Перечислите основные библиотеки Python. Кратко охарактеризуйте каждую их них.
7. Охарактеризуйте дистрибутив Anaconda. Какими свойствами он обладает?
8. Охарактеризуйте интерактивную оболочку Jupyter-notebook. Какими свойствами она обладает?
9. Охарактеризуйте интерактивную оболочку Spyder. Какими свойствами она обладает?
10. Охарактеризуйте интерактивную оболочку PyCharm. Какими свойствами она обладает?
11. Охарактеризуйте интерактивную оболочку RStudio. Какими свойствами она обладает?

Глава 2. ОСНОВНЫЕ СИНТАКСИЧЕСКИЕ КОНСТРУКЦИИ PYTHON

Вся информация, которая хранится в компьютере представляет собой последовательность битов. Каждый бит может принимать два значения 0 или 1. Важнейшим преимуществом вычислительной техники является то, что можно интерпретировать эти биты удобным нам способом. Это могут быть данные различных типов и размера (например, числа или текстовые символы), а может компьютерный код (например, код языка программирования).

Использование языка программирования Python заключается в формировании наборов таких битов, которые бы соответствовали решению конкретных задач пользователя, а также для отправки необходимых данных в процессор компьютера и дальнейшего получения результата.

В данной главе будут рассмотрены типы данных, используемых в Python, и значения, которые они могут содержать.

Программы, использующие Python, получают доступ к определенной области памяти компьютера с помощью операционной системы. Операционная система гарантирует, что программа не может читать или записывать данные в другие области памяти без соответствующего разрешения.

Разные типы данных используют разное количество бит. 64-битная машина означает, что целое число использует 64 бита (8 байт). Некоторые языки программирования хранят необработанные значения в памяти компьютера, отслеживая их размеры и типы. Вместо непосредственной обработки таких данных Python упаковывает каждое значение в память как объект.

Можно представить, что объекты - это коробки переменной длины, занимающие место на полках. Python создает такие коробки, размещает их на свободных местах на полках и убирает, когда потребность в них отпадает. Объектом в Python является фрагмент данных, имеющий как минимум: тип, уникальный идентификатор, значение, соответствующее типу, и счетчик ссылок для отслеживания частоты использования.

С точки зрения компьютера все биты одинаковы. Одни и те же биты могут иметь разные значения в зависимости от типа данных, которыми они являются (например, бит может представлять число 65 или символ A).

В Python **идентификатор** - это имя, использующееся для определения переменной, функции, класса, модуля и других объектов. Объекты в Python представляют собой абстракцию данных. Все данные в программе на Python

представлены объектами или отношениями между объектами. В некотором смысле и в соответствии с моделью фон Неймана о «компьютере с хранимыми программами» код также представлен объектами.

Каждый объект характеризуется идентификатором, типом и значением. Идентификатор объекта является уникальным и постоянным и фактически представляет собой адрес объекта в памяти. Оператор **is()** сравнивает идентификаторы объектов, а функция **id()** возвращает целое число, представляющее идентификатор объекта. Тип объекта определяет операции, которые поддерживает объект, а также возможные значения для объектов этого типа. Функция **type()** возвращает тип объекта, который сам является объектом.

В языке программирования Python тип объекта определяет его свойства, то есть показывает можно ли изменить значение, хранящееся в объекте. Если значение в объекте изменить нельзя, то оно называется неизменяемым, или постоянным. Неизменяемый объект условно можно сравнить с закрытым ящиком с прозрачными стенками - его значение можно увидеть, но нельзя изменить. Изменяемый объект, наоборот, похож на коробку с крышкой - значение объекта можно не только увидеть, но и изменить, сохранив его тип. В Python тип объекта не изменяется, даже если его значение можно поменять.

В таблице 1 представлены основные типы данных в Python.

Таблица 1 – Типы данных в Python

Имя	**Тип**	**Возможность изменения**	**Пример**
Булево значение	bool	нет	True, False
Целое число	int	нет	47, 25000,12
Число с плавающей запятой	float	нет	3.14, 27e5
Комплексное число	complex	нет	3j, 5+9j
Текстовая строка	str	нет	'alas', "alack", '"a verse attack"'
Список	list	да	['Winken', 'Blinken', 'Nod']
Кортеж	tuple	нет	(2, 4, 8)
Байты	bytes	нет	b'ab\xff'
Массив байтов	bytearray	да	bytearray(…)
Множество	set	да	set([3, 5, 7])
Фиксированное множество	frozenset	нет	frozenset(['Elsa', 'Otto'])
Словарь	dict	да	{'game': 'bingo', 'dog': 'dingo', 'drummer': 'Ringo'}

В столбце «Тип» указано название каждого типа данных в Python, а столбец «Возможность изменения» показывает, можно ли изменить значение переменной после её создания. В столбце «Пример» приведены примеры-литералы для каждого типа.

В Python два вида определения данных: как литералы и как переменные. Например, целые числа — это последовательность цифр, а дробные числа содержат десятичную точку. Текстовые строки заключаются в кавычки и так далее.

Переменные являются ключевыми понятиями языков программирования и язык Python не исключение. Он, как и большинство других компьютерных языков позволяет определять переменные - имена для значений в памяти нашего компьютера, которые далее будут использоваться в программе.

Имена переменных в Python подчиняются определённым правилам:

- они могут содержать только следующие символы: буквы в нижнем регистре (от a до z); буквы в верхнем регистре (от A до Z); цифры (от 0 до 9); нижнее подчёркивание (_).
- они чувствительны к регистру: thing, Thing и THING - это разные имена.
- они должны начинаться с буквы или нижнего подчёркивания, но не с цифры.
- Python особым образом обрабатывает имена, которые начинаются с нижнего подчёркивания.
- они не могут совпадать с зарезервированными словами Python - их также называют ключевыми.

Список зарезервированных слов и выполняемых ими функций приведён в таблице 2.

Таблица 2 – Зарезервированные слова в Python

Имя переменной	**Выполняемая функция**
false	Логическое значение, которое представляет собой «ложь»
await	Используется для ожидания выполнения сопрограммы
else	Позволяет определить блок кода, который выполняется, если условие в if не выполнено
import	Позволяет импортировать модули в программу
pass	Оператор-заполнитель, который используется, когда программа требует синтаксически правильного выражения, но этого не требуется
none	Представляет отсутствие значения или объекта
break	Предназначен для досрочного выхода из цикла

except	Определяет блок кода, выполняющийся при возникновении исключения
in	Проверяет, находится ли значение в последовательности (например, списке или строке)
raise	Вызывает исключение
true	Логическое значение «истина»
class	Создаёт новый класс
finally	Определяет блок, который всегда выполняется после завершения цикла или функции, независимо от того, возникло ли исключение или нет
is	Используется в сравнениях, чтобы проверить, указывают ли две ссылки на один и тот же объект
return	Возвращает значение из функции
and	Логический оператор «и»
continue	Переходит к следующей итерации цикла, пропуская оставшийся код в блоке
for	Создаёт цикл с заданным числом итераций
lambda	Создаёт анонимную функцию
try	Запускает блок кода и перехватывает любые исключения, возникающие при выполнении блока
as	Используется при создании псевдонимов или переименовании объектов
def	Определяет функцию
from	Импортирует только определённые атрибуты модуля
nonlocal	Объявление переменной в охватывающей области видимости
while	Создаёт цикл, который повторяется до тех пор, пока определённое условие истинно
assert	Проверяет условие, и, если оно ложно, вызывает ошибку assertionerror
del	Удаляет объекты
global	Делает переменную глобальной из локальной области видимости
not	Логический оператор, инвертирующий значение булева выражения
with	Управляет ресурсами, такими как файлы, контекстуально
async	Позволяет программе выполнять более одной задачи одновременно
elif	Условное выполнение, альтернатива else if
if	Определяет условие для выполнения

2.1 Числа и арифметические операторы Python

В Python числа делятся на категории в зависимости от способа их использования. Основные категории чисел в Python: целые числа **int** (сокращение от **integer**) и вещественные **float** (или числа с плавающей запятой). Чтобы определить тип числа или переменной, можно использовать встроенную функцию **type()**. Для этого необходимо запустить

командную строку и активировать Python. В скобках необходимо ввести число или переменную, для которой необходимо узнать её тип. Пример:

```
>>> type(7)
<class 'int'>
>>> type(15.2)
<class 'float'>
>>> x = 5
>>> type(x)
<class 'int'>
```

И примера видно, что значение 15.2 — это число с плавающей запятой, поэтому Python определяет его тип, как вещественный выводит строку float. Переменная x содержит целое число 5, и Python определяет его тип как int.

Арифметические операции в Python выполняются с помощью соответствующих операторов (таблица 3).

Таблица 3 – Основные арифметические операции в Python

Арифметическая операция	**Оператор**	**Алгебраическое выражение**	**Выражение Python**
Сложение	+	a + b	a + b
Вычитание	-	a - b	a - b
Умножение	*	a · b	a * b
Возведение в степень	**	x**y	a ** b
Деление	/	x / y	x / y
Целочисленное деление	//	[x / y]	x // y
Остаток от деления	%	r mod s	r % s

Операции сложение и вычитание в Python выполняются и записываются так же, как в алгебре. Пример:

```
>>> 5 + 2
7
>>> 5 - 2
3
```

Для выполнения операции умножения используется символ * (звёздочка). Пример:

```
>>> 5 * 5
25
```

Возведение в степень выполняется с помощью двух знаков **. Пример:

```
>>> 3 ** 2
9
>>> 81 **(1/2)
9.0
```

Деление выполняет оператор (/), который делит числитель на знаменатель. Пример:

```
>>> 10 / 2
5.0
>>> 10 / 8
1.25
```

Операция целочисленного деления (//) в Python используется для деления одного целого числа на другое. Результатам деления будет целая часть числа, которая не делится. При этом дробная часть числа отбрасывается. Пример:

```
>>> 10 // 8
1
>>> 20 // 8
2
>>> - 17 // 4
-5
```

В Python для вычисления остатка от деления используется оператор вычисления остатка от деления (%). Пример:

```
>>> 21 % 5
1
>>> 18 % 4
2
>>> 9.5 % 4.5
0.5
```

Также, как и в алгебре, в Python деление на 0 запрещено. При попытке выполнения такой операции происходит исключение, и Python выдаёт трассировку стека. В этой трассировке указано, что произошло исключение типа **ZeroDivisionError**. Zero - это указание на то, какое действие вызвало ошибку. DivisionError – это название ошибки, которая произошла при делении. Большая часть исключений заканчивается суффиксом Error, поэтому по ошибке с этим суффиксом можно понять, что она произошла при выполнении определённой операции. В данном случае суффикс Error в названии исключения указывает на ошибку при делении (division). Например:

```
>>> 10 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

2.2 Строки

Строки в Python представляют собой упорядоченные последовательности символов и используются для хранения и представления текстовой информации. С помощью строк можно работать со всеми данными, представленными в текстовой форме. Отдельного символьного типа в Python нет, символ рассматривается как строка длины 1. Таким образом, в Python символы, являющиеся элементами строки, тоже представляют собой строки.

2.2.1 Создание строк

В Python строка создаётся путём заключения символов в парные одинарные или двойные кавычки. Пример:

```
"Строка Python."
'Строка Python.'
```

2.2.2 Основные операции и функции со строками

Сложение строк. С помощью оператора «+» строки, как и числа, можно складывать, или «склеивать» между собой. Эту операцию еще называют «сцеплением». Пример:

```
a = "Привет"
b = "Солнце"
c = a + b
print(c)
# вывод: "Привет Солнце "
```

Вычитание строк. По аналогии с числами в Python нет возможности выполнять операцию вычитания для строк. Однако с помощью функции **replace()** можно заменить одно слово в строке на другое.

Например, есть строка с текстом «`Я люблю пить кофе по утрам`». Теперь необходимо заменить слово «`кофе`» на «`чай`», используя метод **replace()**. Воспользуемся командой на примере:

```
x = "Я люблю пить кофе по утрам"
x = x.replace("кофе", "чай")
print(x)
# вывод: "Я люблю пить чай по утрам".
```

Стоит отметить, что метод **replace()** заменяет все вхождение подстроки. Если необходимо заменить нужную строку определенное число раз, например, два раза, то третьим параметром нужно подать необходимое число замен. Например:

```
text = "Я люблю кофе и кофе"
text.replace('кофе', 'чай', 1)
# заменит одно вхождение слова "кофе" на "чай"
# вывод: "Я люблю пить чай и кофе".
```

Дублирование (умножение). Строки в Python можно умножать на целые числа, при этом происходит повторение содержимого строки заданное количество раз. Пример:

```
a = "Привет, мир!"
print(a * 3)
```

Этот код выведет строку "**Привет, мир! Привет, мир! Привет, мир!**". Можно указать необходимое количество повторений строки.

Определение длины строки. Функция **len()** - это одна из встроенных функций языка программирования Python. Она позволяет определять длины объекта, например, количества элементов в строке.

Функцию **len()** можно применять к различным типам данных, но не все типы данных могут быть допустимыми аргументами для этой функции. Пример:

```
a = 'привет, мир'
b = [1, 2, 3, 4, 5]
print(len(a))
# выведет 11 (количество символов в строке a)
print(len(b))
# выведет 5 (количество элементов в списке b)
```

Разделение строки. Разделение строки можно выполнить с помощью различных методов, которые зависят от типа данных и желаемого результата. К ним относятся: функция **split()**, функция **rsplit()**, функция **find()**. Эти методы основаны на использование строк, списков и срезов.

Функция **split()** предназначена для разделения строки на массив подстрок по определенному разделителю. Эта функция принимает в качестве аргумента разделитель (например, пробел или запятую) и возвращает список подстрок, каждая из которых представляет отдельный элемент строки после разделения. С её помощью можно также разделить строку по нескольким разделителям или получить подстроки определённой длины.

Функция **rsplit()** работает аналогично функции **split()**, но позволяет указать максимальное количество разделений - разделение будет выполняться справа до указанного лимита. Эту функцию можно использовать для извлечения только интересующих разделов в строках с повторяющимися разделителями.

Другой подход к разделению строки заключается в использовании метода **find()**, который находит индекс первого появления указанного символа (или подстроки) в строке.

Есть и другие методы, предназначенные для извлечения чисел из строки или работа с регулярными выражениями. Выбор метода зависит от конкретной задачи и типа обработки данных, которую необходимо выполнить.

Рассмотрим пример кода, который разделяет строку на слова и выводит их с помощью функции **split()**:

```
stroka = "Как приятно, что вы изучаете Python!"
spisok = stroka.split()
print("Список слов:", spisok)
```

Код выведет: `Список слов: ['Как', 'приятно', 'что', 'вы', 'изучаете', 'Python!']`

Объединение (конкатенация) строк. Python предлагает несколько способов объединения строк, среди которых особенно популярными являются метод **join()** и оператор **+**. Метод **join()** служит для объединения элементов итерируемого объекта, такого как список или кортеж, в единую строку. Этот метод обеспечивает более гибкий подход к конкатенации строк и особенно удобен при работе с множеством строковых элементов. Более того, использование метода **join()** может повысить производительность за счёт снижения количества операций выделения памяти.

Важно отметить, что для успешного выполнения операции объединения содержимое списка должно состоять исключительно из строк. В случае наличия в списке строк чисел или других типов данных после применения метода **join()** при выводе на экран возникнет ошибка **TypeError**. Пример использования функции **join()**:

```
strings = ['Привет', 'Прекрасный, мир!']
separator = ' '
result = separator.join(strings)
print(result)
# выведет: " Привет Прекрасный мир!"
```

Замена символов в строке. Для замены символов в строке используется функция **replace()**. Синтаксис функции выглядит следующим образом: `string.replace(old, new)`, где

string - исходная строка,

old - заменяемый символ,

new - новое значение символа.

Чтобы продемонстрировать работу функции **replace()** рассмотрим следующий пример: предположим, есть строка «Python – популярный язык программирования». Необходимо заменить символ «-» на символ «*». Для этого воспользуемся рассмотренной функцией **replace()**, которая принимает два аргумента - заменяемый символ и символ, которым нужно заменить:

```
text = "Python – популярный язык программирования"
new_text = text.replace('-', '*')
print(new_text)
```

В результате выполнения кода строка будет выглядеть следующим образом: **«Python * популярный язык программирования»**.

Удаление пробельных символов в строке. Функция **strip()** в Python используется для удаления пробельных символов с обоих концов строки.

Синтаксис: `strip([символы])`, где

[символы] - это строка, которая может содержать символы, которые необходимо удалить (по умолчанию удаляются пробелы). Если символы не указаны, функция удаляет все общие пробельные символы.

Функция **strip()** является методом строк и работает с текстовыми переменными. Также у этой функции есть дополнения:

- rstrip() – удаляет указанные или пробельные символы только справа;

- lstrip() – удаляет указанные или пробельные символы слева.

Пример применения функции:

```
a = "  apples and oranges  "
print("Исходная строка:",a)
b = a.strip()
print("После использования метода strip:",b)
c = a.lstrip()
d = a.rstrip()
print("Результат после использования left strip:",c)
```

Результат после использования left strip: apples and oranges

```
print("Резульат после использования right strip:",d)
```

Резульат после использования right strip: apples and oranges

2.3 Циклы

Циклы — это конструкции, которые повторяют определённую последовательность действий многократно. В реальной жизни человек сталкивается с циклами довольно часто: примером может служить ходьба человека, которая представляет собой цикл «левой-правой» до достижения цели (к примеру, университета или кафе).

В компьютерных программах существуют не только инструкции ветвления, подразумевающие выбор пути действия (**конструкция if-else**), но и инструкции циклов, обеспечивающие повторение действий. Без использования инструкций циклов приходилось бы вставлять в программу один и тот же код такое количество раз, которое соответствует необходимому числу повторений этой последовательности действий.

2.3.1 Циклы с условием

Цикл с условием (конструкция while) является универсальным организатором цикла в языке программирования Python (как и во многих других языках программирования). Слово «while» переводится с английского как «пока», что означает, что цикл будет выполняться до тех пор, пока не будет выполнено заданное условие.

Конструкция **while** на языке Python может быть представлена следующим образом:

```
x = 1
while x <= 5:
 print(x)
 x = x + 1
```

Этот цикл будет выполняться, пока значение переменной x меньше или равно 5. В каждой итерации значение x увеличивается на 1, а затем выполняется тело цикла.

Когда переменная превышает заданное значение (в нашем примере 5), условие цикла становится ложным и выполнение цикла прекращается. При выполнении этого кода на экране появится следующий вывод:

```
1
2
3
4
5
```

2.3.2 Цикл обхода последовательности (for)

Цикл for, как правило, используются либо для повторения какой-либо последовательности действий заданное число раз, либо для изменения значения переменной в цикле от некоторого начального значения до некоторого конечного.

Для повторения цикла for заданное количество раз можно использовать его вместе с функцией **range()**:

```
 for i in range(n):
Тело цикла.
```

где n — это числовая переменная, значение которой служит условием для выполнения тела цикла.

Важно понимать, что циклы позволяют автоматизировать процесс повторения действий, что особенно полезно при работе с большим объёмом данных или при необходимости выполнения повторяющихся задач без участия пользователя. Пример:

```
for x in range(1, 6):
  print(x)
```

Этот цикл будет повторять код внутри фигурных скобок (тело цикла) пять раз, каждый раз увеличивая значение итерационной переменной (x = 1, 2, ..., 5). В данном случае для генерации последовательности чисел используется функция **range()**. Результат работы программы:

```
1
2
3
4
5
```

2.3.3 Функции прерывания циклов

Операторы **break** и **continue** применяются в циклах while и for, позволяя соответственно принудительно выходить из цикла или пропускать текущую итерацию. Инструкция **pass**, напротив, используется для игнорирования заданных условий и продолжения работы программы. Рассмотрим применение этих операторов на примерах.

В Python существует инструкция break, которая служит для немедленного завершения цикла. Эта инструкция может использоваться только в циклах for и while. Как только программа обнаруживает команду break, она завершает цикл, после чего начинает выполняться код, следующий за телом цикла.

Действие оператора break в Python приводит к завершению цикла while, даже если условие цикла while равно True, а также цикла for, несмотря на то, что элемент может удовлетворять условию.

Инструкция break обычно используется при последовательном поиске. К примеру, чтобы найти элемент списка, можно написать цикл for. В этом случае в цикл добавляют условие, которое проверяет, найден ли этот элемент. Если нужный элемент найден, программа выходит из цикла, не проверяя остальные элементы списка. Пример использования инструкции break в цикле for:

```
for x in range(6):
  if x == 4:
    break
  print(x)
```

Этот код выведет числа 0, 1, 2 и 3, потому что, когда значение x станет равным 4, оператор **break** завершит цикл.

Рассмотрим пример использования инструкции **continue** с оператором while:

```
count = 0
while count < 5:
     count += 1
     if count == 3:
         continue
     print(count)
```

В этом примере цикл while будет повторяться до тех пор, пока переменная count не станет равной 5. Если переменная count равна 3, то управление будет передано в начало цикла, пропуская оставшиеся команды внутри цикла. В противном случае, если значение переменной count отлично от 3, будет выполнена инструкция print(count). Вывод этого кода будет следующим:

```
1
2
4
```

Рассмотрим пример использования инструкции **pass** с оператором while:

```
 x = 0
 while x < 5:
     pass  # здесь можно разместить код, который будет выполняться в цикле
     x += 1
 print("Цикл завершён.")
```

В данном примере объявляется переменная x, инициализируя её значением 0. Затем, используя оператор while задается цикл, который будет повторяться, пока значение переменной x меньше 5. В теле цикла (после

инструкции pass) можно разместить код, который будет выполнять нужные действия, однако в данном случае ничего не происходит. Когда значение x становится равным 5, цикл завершается и выводится сообщение «Цикл завершён».

Рассмотрим один из примеров использования else с условным оператором if внутри цикла for. Создаётся список чисел и цикл проходит по каждому числу. В зависимости от того, больше ли число, чем 5, значение выводится на экран. Обратите внимание, что инструкция else срабатывает, если условие цикла не вызвало break.

```
numbers = [3, 12, 6, 4, 9]
for num in numbers:
     if num > 5:  # Если число больше 5
         print(f'{num} больше 5')
         break # Окончание цикла if условие выполнено
else:
    print("Ни одно число не больше 5")  # Инструкция `else`, которая срабатывает, когда условие не истинно
```

Этот пример выведет число 12 на экран, так как оно превышает 5. Затем выполнение цикла будет остановлено инструкцией break, а инструкция else: будет проигнорирована.

В результате выполнения программы на экран будет выведено:

```
12 больше 5
```

2.4 Списки

Список - это упорядоченный набор элементов, каждый из которых имеет свой номер, или индекс. При этом первый элемент списка имеет нулевой индекс, второй - первый и так далее.

Основными операциями для работы со списками являются: индексирование, срезы, а также добавление и удаление элементов. Кроме того, с помощью определённых методов можно проверить, содержится ли конкретный элемент в заданной последовательности.

2.4.1 Создание списков

Для создания списков в Python могут использоваться квадратные скобки []. Элементами списка могут быть любые значения (например, целые и действительные числа, строки и даже другие списки). Пример:

```
fruits = ['яблоки', 'апельсины', 'бананы']
print(fruits)
```

Этот код создаёт список с названиями фруктов. Следует обратить внимание на то, что названия заключены в кавычки и разделены запятыми, а сам список помещён между квадратными скобками. Вывод:

```
['яблоки', 'апельсины', 'бананы']
```

Вторым способом создания списков является применение функции **list()**. Функция list() - это встроенная функция Python, которая используется для создания списка из любой последовательности.

```
# Создание пустого списка
my_list = list()
# Добавление элементов в список
my_list.append("apple")
my_list.append(24)  # число тоже можно добавить в список
print(my_list)
```

В результате выполнения кода сформируется список my_list, в который будут добавлены элементы "apple" и 24. В результате выполнения кода, его вывод будет таким:

```
['apple', 24]
```

Списки можно создавать с помощью циклов, общий вид которых записывается следующим образом:

<выражение> **for** <переменная> **in** <последовательность>. Где: **<выражение>** - элемент списка, **<переменная>** - переменная управления циклом, **<последовательность>** - последовательность объектов, в которой **<переменная>** принимает значение каждого элемента поочерёдно.

Рассмотрим пример создания списка с использованием цикла:

```
# создаем пустой список
my_list = []
# используем цикл для добавления элементов в список
for x in range(6):
    my_list.append(x)
print(my_list)
```

В результате выполнения кода будет создан список `my_list`, в который будут заноситься значения от 0 до 5. Вывод программы будет выглядеть так: **`[0, 1, 2, 3, 4, 5].`**

2.4.2 Операции над списками

Вычисление длины списка. Вычислить длину списка можно с помощью функции **len()**. Функция len() возвращает целое число, соответствующее количеству элементов в списке. Пример:

```
x = [1, 2, 3, 4, 5]
print(len(x))
# выведет 5
```

Добавление элемента в конец списка. Чтобы добавить элемент в конец списка в Python можно использовать метод **append()**. Этот метод добавляет элемент в конец существующего списка. Пример:

```
my_list = [1, 2, 3]
new_element = 4
my_list.append(new_element)
print(my_list)
# Вывод: [1, 2, 3, 4]
```

В этом примере к концу списка `my_list`, состоящему из трёх элементов добавляется переменная `new_element`, значение которой равно 4. В итоге получается список, содержащий 4 элемента.

Добавление элемента в произвольную позицию списка. Используя метод **insert()** можно добавлять элемент в определённое место списка по указанному индексу. Пример:

```
list = ['апельсин', 'банан', 'яблоко']
```

Допустим, необходимо вставить элемент «клубника» перед элементом «банан». Для этого нужно указать номер позиции - 1.

```
my_list.insert(1, "клубника")
```

Теперь список будет выглядеть так: `list = ['апельсин','клубника','банан','яблоко']`.

Удаление элемента из списка. Удаление элемента из списка в Python реализуется несколькими способами. Для того, чтобы удалить первый элемент из списка можно использовать функцию **remove(x)**. Пример использования функции:

```
# список
fruits = ['яблоко', 'апельсин', 'банан', 'лимон']
# удаление элемента «банан»
fruits.remove('банан')
print(fruits)
# результат fruits = ['яблоко', 'апельсин',
'лимон']
```

Для того, чтобы удалить из списка элемент с определённым индексом используют функцию **pop(i).** Пример использования функции:

```
fruits = ['яблоки', 'апельсины', 'виноград', 'дыня']
deleted_element = fruits.pop(2)  # удаляет элемент с индексом 2 и присваивает его значение переменной deleted_element
print(fruits)  # выводит оставшийся список
print(deleted_element)  # вывод удалённого элемента
```

Поиск минимального (максимального) элемента в списке. Для поиска минимального элемента в списке Python можно использовать встроенную функции **min()** и **max()**. Алгоритм поиска минимума в списке заключается в следующем: предполагается, что минимальным является первый элемент. Затем поочерёдно перебираются все элементы списка, начиная со второго. Каждый элемент сравнивается с предполагаемым минимальным, и если находится элемент меньше минимального, то он перемещается на место минимального элемента. Для поиска максимального элемента в списке алгоритм аналогичен. Пример:

```
my_list = [4, 2, 9, 1, 5]
min_element = min(my_list)
print("Минимальный элемент:", min_element)
```

Минимальный элемент: 1

В этом примере создается список, к которому применена функция min(). С ее помощью осуществляется поиск минимального элемента списка. Результат поиска выводится на экран. Функция min() работает с разными типами данных и с ее помощью будет осуществляться поиск минимального элемента среди них.

Необходимо отметить, что функция min() сравнивает элементы списка на основе их значений, а не индексов. В результате она может вернуть значение, которое имеет наименьшее значение среди всех элементов списка, независимо от его положения.

Поиск указанного элемента в списке. Для того, чтобы найти элемент в списке необходимо использовать специальный алгоритм. Он заключается в следующем: изначально предполагается, что искомого элемента нет в списке. Используя цикл последовательно проверяется каждый элемент списка на соответствие искомому элементу. Если совпадение обнаружено, то процесс проверки останавливается. Пример нахождения элемента в списке:

```
list_of_elements = ["яблоко", "груша", "апельсин", "лимон", "персик", "банан"]
item = "апельсин"
```

```
if item in list_of_elements:
    print("Элемент найден")
else:
    print("Элемент не найден")
```

Этот код создает список и определяет элемент, который нужно в нем искать. Затем используя оператор in, проверяется наличие необходимого элемента в списке. Если элемент найден, выдается сообщение «Элемент найден», иначе - «Элемент не найден».

Сортировка списка. Существует несколько методов сортировки элементов списка, один из них это сортировка перестановкой - метод **sort()**. Сортировка данным методом заключается в последовательном сравнении каждого элемента с каждым последующим элементом. Если требуется, элементы меняются местами. Этот способ сортировки довольно медленный, но он наиболее очевидный.

```
my_list = [3, -5, 2, 4, 1]
my_list.sort()
print(my_list)
```

В результате выполнения этого кода список my_list будет отсортирован по возрастанию, и на экран выведется результат: **[-5, 1, 2, 3, 4]**.

Для того, чтобы отсортировать список по убыванию необходимо воспользоваться функцией **sort(reverse=True)**. Пример:

```
my_list = [8, 4, 2, 9, 6]
my_list.sort(reverse=True)
print(my_list)
```

Вывод:

```
[9, 8, 6, 4, 2]
```

2.5 Многомерные списки

В Python, помимо стандартных одномерных списков, существуют многомерные. Стоить отметить, что чаще всего используются двумерные списки.

Многомерные или вложенные списки могут содержать в качестве элементов не только другие списки, но и кортежи, словари и прочие типы данных. Многомерные списки создаются следующим образом:

```
a = [[], [], [], ..., []]
```

Пример задания многомерного списка Python: `thislist = [['Москва', 'Россия'], ['Париж', 'Франция'], ['Нью-Йорк', 'США']]`

В данном примере каждый элемент основного списка представляет собой список, состоящий из двух элементов: названия города и страны.

В многомерных списках возможно обратится к определенному элементу. Для того чтобы это сделать, необходимо указать индексы всех вложенных списков, разделяя их знаком [ ]. Первый индекс указывает на номер строки, второй - на номер столбца. Таким образом, `matrix[i][j]` обращается к элементу `j` списка `i`. Пример:

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
# Обращение к первому элементу второго вложенного списка (матрицы)
print(matrix[1][0])
Вывод: 4
# Обращаемся к третьему элементу первого вложенного листинга
print(matrix[0][2])
Вывод:3
```

2.5.1 Кортежи

Кортеж - это упорядоченный набор фиксированного числа значений, которые нельзя изменять.

По своей структуре кортежи напоминают списки, но есть важное отличие: элементы кортежа невозможно поменять, а для их создания используются круглые, а не квадратные скобки. Плюс кортежей в сравнении со списками в том, что они занимают меньше места в памяти.

Для создания кортежа можно просто присвоить элементы, разделённые запятыми, или указать их в скобках через запятую. Например:

```
thistuple = ("apple", "banana", 1, 2.5)
```

В данном примере `thistuple` — это кортеж, который состоит из строки `"apple"`, `"banana"`, целого числа `1` и числа с плавающей точкой `2.5`.

В отношении кортежей применяются все операции, которые предусмотрены для списков, при этом изменения элементов не происходит. Это операции: **len()**, **in**, **max()** и другие. Рассмотрим пример использования функции len():

```
my_tuple = ("apple", "banana", "cherry")
len(my_tuple)
```

В рассматриваемом примере `my_tuple` — это кортеж из трёх элементов, и функция `len(my_tuple)` выведет значение 3 - число элементов в кортеже.

Функция zip() объединяет элементы итерируемых последовательностей в кортежи по определённому принципу. Происходит это следующим образом: первые элементы всех последовательностей объединяются в первый кортеж, вторые элементы - во второй кортеж и так далее. Процесс останавливается, когда доходит до конца самой короткой последовательности.

Чаще всего для удобства дальнейшей работы полученные кортежи собирают в одну последовательность. В зависимости от задачи это может быть список, кортеж, словарь или множество. Последовательности, которые используются в качестве аргументов в функции zip() могут быть разных типов. Рассмотрим пример использования функции zip() для объединения элементов двух кортежей в один:

```
# Объединяем кортеж с именами и кортеж с возрастами
names = ('Иван', 'Мария', 'Сергей')
ages = (25, 30, 27)
# Функция zip объединяет два списка в список пар
name_and_age = zip(names, ages)
print(name_and_age)
```

В данном примере результатом работы функции print() будет:

```
[('Иван', 25), ('Мария', 30), ('Сергей', 27)]
```

Функция enumerate() используется для итерируемых последовательностей, когда необходимо не только получить доступ к элементам этой последовательности, но и узнать их индексы. В результате работы функции образуется кортеж, который содержит два элемента: индекс элемента и его значение. Рассмотрим пример использования функции:

```
fruits = ['яблоко', 'груша', 'апельсин', 'банан']
for i, fruit in enumerate(fruits):
    print(i, fruit)
```

В этом примере создается список фруктов и используеться функция **enumerate()**. В результате получим вывод названий фруктов и их порядковых номеров. Вывод:

```
0 яблоко
1 груша
2 апельсин
3 банан
```

2.5.2 Словари

Словарь **dict** - это один из способов реализации ассоциативного массива, который позволяет хранить коллекции элементов в виде пар

«**ключ - значение**». В словарях ключи должны быть уникальными, а значения могут совпадать и быть данными любого типа.

В языке программирования Python словари основаны на **хеш-таблицах** - структурах данных, которые позволяют быстро выполнять операции по добавлению, удалению и поиску элементов. Ключами в словаре могут быть только хешируемые объекты - объекты, которые нельзя изменить. Хешируемыми, как правило, являются строки, кортежи и числа, но не списки. Размер словаря не нужно определять заранее, он автоматически расширяется при добавлении новых элементов.

Значения в словаре могут быть любого типа данных: числа, строки, списки, кортежи и другие словари. Стоит отметить, что в словарях элементы располагаются в произвольном порядке: изменение, удаление или добавление элементов может привести к изменению порядка.

Словари итерируемы: элементы можно перебирать с помощью циклов, например, for. Если перебор значений идёт в прямом порядке, то перебираются ключи, если в обратном - значения. Можно возвращать пары «ключ - значение», это зависит от метода итерации. Также словари часто выступают в роли хранилища данных. Объёмы данных могут быть значительными размеров, но благодаря уникальности ключей доступ к ним остаётся быстрым.

Словарь в Python можно создать с помощью фигурных скобок { } или функции **dict()**. Пример создания словаря:

```
my_dict = {'яблоко': 5, 'апельсин': 3, 'груша': 7}
```

В данном словаре существует три элемента, представляющие собой ключи и соответствующие им значения: «яблоко»: 5, «апельсин»: 3 и «груша»: 7. Пары «**ключ- значение**» разделены запятыми.

Преобразование списков и кортежей в словарь. Преобразовать список в словарь - значит представить список, как набор пар «ключ-значение», где каждый элемент списка становится либо ключом, либо значением. Например, есть список, содержащий имена людей и их возраст (в годах):

```
list_of_people = ['Иван', 'Мария', 'Пётр', 'Евгений']
ages = [23, 37, 49, 25]
```

Для того, чтобы преобразовать этот список элементов в словарь, необходимо воспользоваться функцией dict(). Эта функция использует два аргумента: последовательность ключей и последовательность значений. В рассматриваемом примере ключи являются именами людей, а значения – их возрастом:

```
people_dict = dict(zip(list_of_people, ages))
```

Функция **zip()** объединяет элементы двух списков по индексам, чтобы можно было создать пары «ключ - значение», а затем передать эти пары функции dict(), которая создаст словарь из полученных пар. Таким образом, формируется словарь `people_dict` будет выглядеть следующим образом:

```
people_dict = ['Иван': 23, 'Мария': 37, 'Пётр': 49,
'Евгений': 25]
```

Теперь можно легко найти возраст любого человека по его имени, используя словарь: например, в этом словаре наглядно видно, что человеку по имени Евгений 25 лет.

2.6 Файлы и исключения

При обработке данных возникает необходимость работы с файлами, анализа больших объёмов данных, сохранения пользовательских данные, чтобы они не потерялись после завершения работы программы. Кроме того, при работе с файлами важно научиться обрабатывать ошибки, чтобы избежать аварийного завершения программы. В языке программирования Python для управления ошибками существуют специальные объекты - исключения.

2.6.1 Работа с файлами

Файл - это совокупность данных, представленная в виде последовательности битов и сохранённая на компьютере. Эти данные, хранящиеся в определённой структуре, имеют название «имя файла» (filename).

В языке программирования Python файлы подразделяются на два типа: текстовые и бинарные. К текстовым файлам относятся любые файлы, сохраняющие информацию в текстовом виде. Текст может быть сохранён в двух форматах: в формате простого текста (.txt) и в формате обогащённого текста (.rtf).

Данные в бинарных файлах отображаются в закодированном виде, представляя собой последовательности битов, в которых информация представлена только с помощью нулей и единиц, заменяющих привычные символы. Эти файлы имеют формат (.bin). К ним относятся изображения, аудио и видеофайлы.

При работе с файлами необходимо последовательно выполнить ряд операций: открыть файл с помощью метода **open()**, затем, в зависимости от задачи, прочитать файл при помощи метода **read()** или записать в него

данные с помощью метода **write()**, после чего обязательно закрыть файл методом **close()**.

Открытие файла. Функция **open()** в Python создаёт объект с методами и атрибутами, которые можно использовать для управления файлом. С помощью этого объекта можно производить чтение из файла, запись в файл и изменение файла.

```
f = open(file_name, access_mode)
```

file_name - это имя открываемого файла, а **access_mode** - режим открытия файла, который может быть установлен для чтения (r), записи (w), добавления (a), чтения и записи (r+), а также, если файл уже существует и нужно его изменить, w+ , для дозаписывания в файл - a+ . По умолчанию применяется режим r, если не указано иное. Режимы открытия файлов представлены в таблице 4.

Таблица 4 – Режимы открытия файла

Режим	**Описание**
Текстовые файлы	
r (Read)	Открытие файла только для чтения.
w (Write)	Открытие файла только для записи (создаст новый файл, если не найдет с указанным именем).
a (Append)	Открытие файла для добавления нового содержимого (создаст новый файл для записи, если не найдет с указанным именем).
r+	Открытие файла для чтения и записи.
w+	Открытие файла для чтения и записи (создаст новый файл для записи, если не найдет с указанным именем).
a+	Открытие файла для добавления нового содержимого (создаст новый файл для чтения записи, если не найдет с указанным именем).
Бинарные файлы	
rb	Открытие файла только для чтения (бинарный).
wb	Открытие файла только для записи (бинарный) (создаст новый файл, если не найдет с указанным именем).
ab	Открытие файла для добавления нового содержимого (бинарный). (создаст новый файл для записи, если не найдет с указанным именем).
rb+	Открытие файла для чтения и записи (бинарный).
wb+	Открытие файла для чтения и записи (бинарный) (создаст новый файл для записи, если не найдет с указанным именем).
ab+	Открытие файла для добавления нового содержимого (бинарный). (создаст новый файл для чтения записи, если не найдет с указанным именем).

Пример открытия текстового файла:

```
f = open('my_file.txt')
```

В данном примере режим открытия не указан. Файл откроется по умолчанию в режиме чтения- **r (Read)**.

```
f = open('my_file.txt','r+')
```

В следующем примере используется режим r+. Файл откроется для чтения и записи.

Закрытие файла. После открытия файла в Python необходимо его закрыть. Python автоматически закрывает файл, когда присваивается его объект другому файлу. Существуют два способа закрытия файлов. Проще всего после открытия файла закрыть его, применив метод **close()**.

После закрытия этот файл нельзя будет использовать до тех пор, пока заново его не открыть.

Чтение и запись файлов в Python. Функция **read()** используется для чтения данных из текстового файла. Она возвращает считываемые данные в виде строки или байтового объекта, в зависимости от режима открытия файла. Синтаксис:

```
file.read(size)
```

где **file** -открытый файл; **size** -длина считываемых данных. Это необязательный аргумент, по умолчанию равен -1, что означает считывание всех доступных данных.

Функция возвращает прочитанные данные. Если достигнут конец файла, функция вернёт пустую строку.

Функция **write()**используется для записи указанной строки в файл. Она принимает в качестве аргумента строку, которую требуется записать, и записывает её в файл, на который указывает объект файла. Синтаксис функции write():

```
file_object.write(string)
```

где **file_object** -это файловый объект, ассоциированный с файлом, в который производится запись; **string** -строка, которая будет записана в файл.

Функция write() не добавляет автоматически символ новой строки в конец записанной строки, это следует делать самостоятельно.

2.6.2 Ошибки и исключения

Существует как минимум два вида ошибок: синтаксические ошибки и исключения. Синтаксические ошибки, также известные как ошибки синтаксического анализа, являются, пожалуй, самым распространённым видом проблем, с которыми можно столкнуться при работе с Python. Пример:

```
'Hello world' ( printTruewhile>>>
 Файл "<stdin>", строка 1
    при True print('Hello world')
SyntaxError: недопустимый синтаксис
```

В рассматриваемом случае ошибка произошла в функции print(), поскольку перед ней нет двоеточия (':'). Также выводятся имя файла и номер строки, чтобы можно было легко определить местоположение ошибки, если данные поступают из скрипта.

Даже если оператор или выражение синтаксически корректны, при попытке их выполнения могут возникать ошибки. Ошибки, которые обнаруживаются во время выполнения программы, называются исключениями. Они не являются безусловно фатальными. Тем не менее, большинство исключений не обрабатываются программами, что приводит к появлению сообщений об ошибках, как показано ниже.

```
Файл "<stdin>", строка 1, в <модуле>
ZeroDivisionError: деление на ноль
4 + спам*3
Трассировки (последний вызов):
 Файл "<stdin>", строка 1, в <модуле>
NameError: имя 'spam' не определено
'2' + 2
трассировки (последний вызов):
 Файл "<stdin>", строка 1, в <модуле>
TypeError: может объединять только str (не "int") в
str
```

В сообщении об ошибке последняя строка указывает на причину возникновения ошибки. Исключения бывают разных типов, и тип ошибки печатается как часть сообщения. В приведённом примере типами исключений являются **ZeroDivisionError**, **NameError** и **TypeError**. Строка, указывающая на тип исключения, соответствует названию возникшего встроенного исключения. Это характерно для всех встроенных исключений, но необязательно для пользовательских, хотя это может быть полезным правилом. Стандартные имена исключений представляют собой встроенные идентификаторы (не зарезервированные ключевые слова).

В остальной части строки содержатся подробные сведения о типе исключения и о том, что его вызвало.

Перед этим в сообщении об ошибке отображается контекст, в котором произошло исключение, в виде обратной трассировки стека. Как правило, она содержит список строк исходного кода, но не будет отображать строки, считанные со стандартного ввода.

Во встроенных исключениях перечислены встроенные исключения и их значения, это помогает определить причину возникновения ошибки и устранить её.

Контрольные вопросы по главе 2

1. Перечислите типы данных, используемые в Python.
2. Перечислите зарезервированные слова в Python и выполняемые ими функции.
3. Каким образом в Python создаются строки?
4. Какие операции можно выполнять со строками. Перечислите используемые при этом функции Python.
5. Какие виды циклов вы знаете? Дайте им краткую характеристику.
6. Перечислите конструкции, используемые для задания циклов в Python.
7. Каким образом в Python создаются списки?
8. Перечислите конструкции, используемые для задания одномерных списков в Python.
9. Перечислите конструкции, используемые для задания многомерных списков в Python.
10. Какие виды файлов используются в Python?
11. Какие операции можно выполнять над файлами в Python?
12. Какие режимы открытия файлов вы знаете?

Глава 3. БАЗОВЫЕ ТЕХНОЛОГИИ ДЛЯ АНАЛИЗА ДАННЫХ

3.1 Возможности библиотеки Pandas

Pandas - это один из самых важных пакетов, который используется при изучении Python. Он известен своей удобной структурой данных *Pandas DataFrame*, кроме того он позволяет с лёгкостью работать с табличными данными такими, как электронные таблицы внутри скрипта на Python.

Pandas является библиотекой с открытым исходным кодом, что позволяет каждому желающему просматривать исходный код и предлагать свои улучшения через запросы на извлечение.

Pandas был создан для работы с двумерными данными, схожими с электронными таблицами Excel. Аналогично тому, как библиотека NumPy имеет встроенную структуру данных, называемую *array*, со специальными атрибутами и методами, библиотека Pandas имеет встроенную двумерную структуру данных под названием *DataFrame*.

Библиотека Pandas включает в себя три основные структуры данных:

- **Series**. Представляет собой одномерный массив фиксированной длины, предназначенный для хранения данных одного типа, что делает его похожим на колонку в таблице.
- **DataFrames**. Образуют двумерные таблицы с возможностью изменения размеров, где каждый столбец может содержать данные разных типов, обеспечивая тем самым гибкость в работе с табличными данными.
- **Panel**. Является трехмерной структурой данных, позволяющей работать с данными, организованными в трех измерениях, обладая при этом изменяемыми размерами.

Объект Series в Pandas представляет собой структуру данных, аналогичную одномерному массиву, который включает в себя ряд значений. Эти значения могут быть различных типов, подобных тем, что используются в NumPy. Каждому элементу в Series соответствует уникальная метка, известная как индекс, что позволяет связать каждое значение с определенной меткой данных.

Чтобы создать объект Series необходимо использовать конструктор **Series()** из библиотеки Pandas. В качестве аргумента в этот конструктор передается массив с данными, которые необходимо включить в создаваемый объект Series. Этот процесс инициализирует Series с указанными значениями, автоматически присваивая индекс каждому элементу, начиная с нуля, если явно не задан другой индекс. Пример создания объекта Series:

```
import pandas as pd
# Создание объекта Series
s = pd.Series([5, -10, 15, -20, 25])
s
0     5
1   -10
2    15
3   -20
4    25
dtype: int64
```

Каждому значению в этом объекте Series присвоен автоматический индекс, начиная с 0, что позволяет легко обращаться к каждому элементу по его индексу.

DataFrame - это основной тип данных в библиотеке Pandas, который представляет собой двумерную структуру данных, напоминающую таблицу с произвольным количеством строк и столбцов. DataFrame позволяет работать с данными разных типов: числовыми, булевыми, строковыми и другими.

Важной особенностью DataFrame являются индексы строк и индексы столбцов. Они упрощают сортировку и фильтрацию данных, а также позволяют быстро получать доступ к определённым ячейкам.

Рассмотрим пример создания простого DataFrame с помощью словаря и осуществим его вывод на экран:

```
import pandas as pd # Импортируется библиотека Pandas.
city = {'Город': ['Москва', 'Санкт-Петербург', 'Новосибирск', 'Екатеринбург'],
        'Год основания': [1147, 1703, 1893, 1723],
        'Население': [11.9, 4.9, 1.5, 1.4]} # Создаём словарь с нужной информацией о городах.
df = pd.DataFrame(city) # Преобразование словаря в DataFrame, используя стандартный метод библиотеки.
df # Вывод DataFrame на экран.
            Город  Год основания  Население
0          Москва           1147       11.9
1  Санкт-Петербург           1703        4.9
2      Новосибирск           1893        1.5
3     Екатеринбург           1723        1.4
```

Pandas предоставляет возможность импортировать данные из различных источников, таких как словари, списки и кортежи. Одним из наиболее распространённых способов является работа с файлами формата *.csv*, которые часто используются в анализе данных. Для этого используется команда **pd.read_csv()**.

Эта команда имеет несколько параметров для настройки процесса импорта.

Параметр sep позволяет указать разделитель, который используется в импортируемом файле. По умолчанию его значение равно запятой (,)**,** что соответствует разделителю данных в формате .csv. Использование этого параметра полезно, когда в исходном файле используется нестандартный разделитель, например, табуляция или точка с запятой.

Параметр dtype позволяет задать тип данных для столбцов после загрузки файла формата .csv. Это полезно в случаях, когда формат данных был автоматически определён неверно. Например, даты часто импортируются как строковые переменные, хотя для них существует отдельный тип данных.

3.2 Возможности библиотеки NumPy

Библиотека NumPy для языка программирования Python играет ключевую роль в анализе данных, машинном обучении и научных вычислениях, а также значительно упрощает работу с векторами и матрицами. Многие библиотеки Python, такие как Scikit-learn, SciPy, Pandas и Tensorflow, используют NumPy как основной элемент своей инфраструктуры.

Владение инструментами NumPy, помимо возможности детализированного анализа числовых данных, даёт значительное преимущество при отладке более сложных сценариев библиотек. Основная задача библиотеки NumPy - работа с массивами.

1. Создание массивов NumPy. Для создания массива необходимо импортировать модуль numpy. Массивы могут быть одномерными и многомерными, отличаться по размеру и иметь разные типы данных в каждом элементе. Существует множество функций для создания массивов:

Функция **np.array()** - позволяет создать новый массив из вложенных последовательностей любого типа данных. Для наглядного понимания и примера создания двумерного массива с помощью функции **np.array()** рассмотрим примеры создания массивов разного размера:

Одномерный массив NumPy.

```
import numpy as np
a = np.array([1,2,3])
print(a)
```

Результатом будет **[1 2 3].**

Этот код создаёт одномерный массив размером 1x3 с элементами 1, 2 и 3.

Многомерные массивы.

```
import numpy as np
arr = np.array([[1,2,3],[4,5,6]])
print(arr)
```

Результат работы программы:

[[1, 2, 3], [4, 5, 6]]

Этот код создаёт двумерный массив размером 2x3 с элементами 1, 2 и 3 в первой строке и элементами 4, 5 и 6 во второй строке. Этот код можно изменить, чтобы создать массив необходимой размерности и с нужными данными.

Функция **np.arange()**. Эта функция, как и функция **range()** создает последовательность элементов.

Рассмотрим пример использования функции **np.arange()** для создания массива чисел:

```
import numpy as np
arr = np.arange(11)
print(arr)
```

Результат работы программы:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

В этом примере импортируется модуль *numpy*, затем ему присваивается псевдоним *arr.* Далее применяется функция **np.arange()**, используя в качестве аргумента число 11. Функция возвращает массив из 11 чисел, начиная с нуля и заканчивая числом 10.

Функции **np.zeros()** и **np.zeros_like()** создают массив нулей заданной формы и размерности. Пример использования функции **np.zeros()**:

```
import numpy as np
arr = np.zeros(5)
print(arr)
```

Результатом выполнения этого кода будет:

[0, 0, 0, 0, 0]

Пример использования функции **np.zeros_like()**:

```
import numpy as np
```

```
x = np.array([1, 2, 3, 4, 5])
y = np.zeros_like(x)
print(y)
```

Функция **zeros_like** возвращает массив с теми же пятью значениями, но уже нулевыми. Вывод программы будет следующим:

```
[0, 0, 0, 0, 0]
```

Функции **np.ones()** и **np.ones_like()** - создают массив единиц заданной формы и размерности. Пример:

```
import numpy as np
x = np.ones(5)
print(x)
```

В результате выполнения этого кода будет выведен массив `[1. 1. 1. 1. 1.]`, состоящий из пяти элементов.

Одна из функций numpy - функция **ones_like**. С её помощью можно создать массив единиц, размер которого совпадает с заданным массивом. Пример:

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
y = np.ones_like(x)
print(y)
```

В результате выполнения этого кода в консоль будет выведена следующая строка:

```
[1. 1. 1. 1. 1.]
```

Функция подключения генератора случайных чисел **np.random()**. Пример использования функции:

```
import numpy as np
np.random(10)
```

В результате выполнения кода будет создан массив случайных чисел от 0 до 9. Значение 10 в аргументе функции определяет верхнюю границу случайного числа.

Функция **np.append()**. Предназначена для добавления элемента в конец списка. Например, необходимо добавить число 5 к имеющемуся списку [1, 2, 3, 4]. Тогда команда будет выглядеть так:

```
import numpy as np
a = np.array([1, 2 ,3, 4])
a = a.append(5)
print(a)
```

В результате в консоль будет выведено следующее: **`[1, 2, 3, 4, 5]`**

2. Арифметические операции над массивами NumPy.

Арифметические операции над массивами в NumPy облегчают работу с большими наборами данных и обеспечивают высокую производительность вычислений.

Сложение. С помощью арифметического оператора «+» можно складывать два массива одинаковой формы. Например:

```
import numpy as np
arr_1 = np.array([1, 2, 3])
arr_2 = np.arange(3)
# сложение элементов массивов arr_1 и arr_2
result = arr_1 + arr_2
print(result)
# Вывод [1 3 5]
```

При сложении двух массивов, размерность которых отличается, библиотека Numpy автоматически соотносит или «исправляет» размерности таким образом, чтобы поэлементная операция стала возможна.

Вычитание. Вычитание осуществляется аналогичным образом:

```
import numpy as np
arr_1  = np.array([1, 2, 3, 4, 5])
arr_2  = np.array([4, 3, -2, 0, 1])
result = arr_1 - arr_2
print(result)
# Вывод [ -3  -1  5  4  4]
```

В результате операции каждый элемент массива **arr_1**, представляющий собой целочисленное значение, уменьшается на соответствующее ему значение из массива **arr_2**.

Умножение и деление. В NumPy есть функция **np.multiply()**, которая выполняет поэлементное умножение двух массивов одинаковой формы. Пример:

```
 import numpy as np
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
result = np.multiply(array1, array2)
print(result)
# Вывод [ 4  10  18]
```

В результате выполнения кода создается два массива NumPy array1 и array2. С помощью функции np.multiply() осуществляется поэлементное умножение двух массивов, с выводом результата - result.

Следующий код демонстрирует операцию деления в NumPy:

```
import numpy as np
array1 = np.array([4, 9, 16])
```

```
array2 = np.arange(3)
result = array1 / array2
print(result)
# Вывод [0 9 8]
```

В этом примере создаются два массива array1 и array2. Затем выполняется операция деления каждого элемента в array1 на соответствующий элемент в array2, в результате чего получается новый массив result.

3. Индексация массива NumPy. Массив NumPy можно разделить на части и каждой присвоить свой индекс. Это работает по аналогии с тем, как это происходит со списками Python. Для индексации массива NumPy необходимо знать основные правила синтаксиса. Если есть массив *a*, то для обращения к одному элементу или группе элементов, можно использовать квадратные скобки ([]), в которых указать требуемые индексы. Например:

- **a [0]** - обращается к первому элементу массива.

- **a [-1]** - к последнему элементу массива.

Если необходимо обратиться к нескольким элементам, то нужно указать диапазон индексов в квадратных скобках через запятую. Например, **a [1:3]** обращается ко второму и третьему элементам массива. Индексы указываются включительно: первый указанный индекс будет включён в результат, а второй - исключён. Если важно получить третий элемент массива с индексом 2, но не нужен второй элемент, указать диапазон a [2:3].

Индексы с отрицательными значениями отсчитываются относительно конца массива. Например:

- **a[-2]** обозначает предпоследний элемент.

- **a[-4:-1]** обращается с четвёртого по первый элементы массива (исключая первый).

Можно также использовать логические индексы для обращения к определённому элементу. С помощью логических индексов можно выбрать элементы массива или списка, основанные на определённых критериях. Это широко используется в различных алгоритмах и анализах данных. Например, логические индексы в Python используют следующим образом:

Допустим, есть массив *a* с некоторыми значениями:

```
a = [1, 2, 3, 4, 5]
```

Чтобы обратиться к определённому элементу массива (например, ко второму), необходимо применить логический индекс. Если нужно получить значение второго элемента (то есть элемента с индексом 1) выполняется следующее действие:

```
element_2 = a[1]
```
 # получается значение 2
```
# вывод значения element_2 с помощью функции print
print(element_2)
```
 # вывод 2

Кроме того, логические индексы можно применять, если необходимо изменить значение элемента массива. Предположим, нужно заменить значение второго элемента массива `a = [1, 2, 3, 4, 5]` на число 9, тогда: `a[1] = 9`

Теперь массив a будет выглядеть так:

[1, 9, 3, 4, 5]

Следует обратить внимание, что логический индекс может быть использован для массива любой размерности. Однако синтаксис будет немного отличаться.

4. Агрегирование в NumPy. Функции агрегирования помогают получать итоговые значения из списков и коллекций в Python. С их помощью можно, например, рассчитать среднее значение, подсчитать количество элементов, получить сумму или найти максимальное и минимальное числа. Рассмотрим пример использования функций **min()**, **max()** и **sum()** в Python:

Предположим, есть список чисел, и необходимо найти минимальное, максимальное число и сумму всех чисел. Применяем к нему указанные функции:

```
my_list = [5, 10, -1, 8, 7, -2]
# Нахождение минимального числа
min_num = min(my_list)
# Вывод минимального числа на экран
print("Минимальное число:", min_num)
#Нахождение максимального числа
max_num = max(my_list)
#Вывод максимального числа на экран
print("Максимальное число:", max_num)
#Считаем сумму чисел в списке
sum_num = sum(my_list)
#Вывод суммы чисел на экран
print("Сумма чисел:", sum_num)
```

Результат выполнения данной программы:

```
Минимальное число: -2
Максимальное число: 10
```

```
Сумма чисел: 21
```

Для вычисления среднего арифметического значения элементов массива необходимо использовать функцию **mean()**. Рассмотрим пример использования данной функции в Python:

```
import numpy as np
arr = [1, 2, 3, 4, 0]
print(np.mean(arr))
```

Этот код выведет среднее арифметическое значений массива arr:

```
5
```

Для вычисления произведения элементов массива необходимо использовать функцию **prod()**. Пример использования функции **prod()** для получения произведения элементов массива:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
product = np.prod(arr)  # вычисляется произведение всех элементов массива
print(product)
# результат: 120
```

5. Создание матриц NumPy. Для создания матриц в библиотеке NumPy можно использовать разные методы. Рассмотрим пример кода на Python, который создает матрицу 3x3:

```
import numpy as np
# Создание матрицы
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matrix)
```

Вывод:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

6. Арифметические операции над матрицами NumPy. В библиотеке NumPy можно выполнять различные арифметические операции с матрицами, такие как сложение, вычитание и умножение. Важно отметить, сложение матриц в NumPy работает только для матриц одинакового размера. При сложении матриц выполняется поэлементное сложение, то есть каждый элемент матрицы-результата будет равен сумме соответствующих элементов матриц-слагаемых. Пример сложения двух квадратных матриц:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array ([[2, 1], [0, 5]])
```

```
result = a + b
print(result)
```

В результате выполнения данного кода создаются две квадратные матрицы *a* и *b* размерами 2x2 и вычисляется их сумма. Результат сложения выводится в следующем виде:

```
[[3 3]
[3 9]]
```

Вычитание матриц происходит по такому же принципу. Пример вычитания:

```
import numpy as np
A = np.array ([[1, 2], [3, 4]])
B = np.array ([[2, -1], [-3, 1]])
print (A - B)
```

Результат выполнения программы:

```
[[-1  3]
 [ 0  5]]
```

Для того чтобы выполнить умножение матриц важно следовать определённым правилам:

- Элементы первой строки первого массива последовательно умножаются на элементы первого столбца второго массива.
- Полученные результаты суммируются, и получается первый элемент результирующего массива.
- Далее процесс повторяется, но уже для следующих элементов обоих массивов, стоящих в тех же позициях.

Рассмотрим пример умножения матриц в бибилиотеке NumPy:

```
import numpy as np
# Создание двух матриц
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
# Умножение матриц (по умолчанию, умножение матричное)
c = a @ b
print(c)
# [19 22; 43 50]
```

7. Индексация матриц NumPy. Индексация матрицы в NumPy используется для доступа к ее элементам. Индексация в библиотеке NumPy соответствует общепринятой нотации, используемой в математике, с одним исключением: вместо одного индекса, как в других языках программирования, используются два индекса.

Например, чтобы обратиться к некоторому элементу матрицы A, нужно указать два индекса: первый определяет номер строки, а второй - номер столбца. В результате получается значение элемента на их пересечении.

Синтаксис для обращения к элементу выглядит следующим образом:

```
A[строка, столбец]
```

Чтобы обратиться ко всему элементу в строке или столбце матрицы, достаточно задать один индекс. Для обращения сразу к нескольким элементам можно задать диапазоны индексов.

Для того чтобы выполнять эффективную индексацию матрицы NumPy, необходимо чётко понимать её структуру и размерность. Для выполнения индексации матрицы в NumPy используются квадратные скобки []. Пример:

```
import numpy as np
# Создание матрицы 3x3
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Вывод второго элемента второй строки:
print(matrix[1][1])
```

Вывод: **5**

Индексация матрицы в NumPy начинается с 0. Поэтому вторая строка - это строка с индексом 1, а второй элемент этой строки - элемент с индексом [1][1]. В результате индексации созданной матрицы результатом будет значение 5.

8. Агрегирование матриц NumPy. Агрегирование (или свёртка) матриц использует различные функции для объединения элементов матрицы или массива. Результат агрегирования часто представляет собой одно значение. Например, нахождение суммы всех элементов в матрице является агрегированием, так как значения элементов сворачиваются в одну общую сумму. Свёртка может быть выполнена по строкам, столбцам или другим измерениям матрицы.

Во время агрегирования применяются различные операции, которые вычисляют сумму, среднее значение, максимум, минимум и другие. В библиотеке NumPy агрегирование осуществляется с помощью таких функций, как **np.sum()**, **np.mean()**, **np.max()** и прочих. Они могут применяться к строкам, столбцам, а также всему массиву или матрице, в зависимости от конкретной задачи и структуры данных.

Для объединения двух матриц одинаковой размерности используется функция **numpy.add()**. Функция принимает две матрицы в качестве аргументов и возвращает матрицу, элементы которой равны суммам соответствующих элементов входных матриц. Пример:

```
import numpy as np
matrix1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
matrix2 = np.array([[9,8,7],[6,5,4],[3,2,1]])
result = np.add(matrix1, matrix2)
print(result)
```
Вывод:
```
[[10 10 10]
 [10 10 10]
 [10 10 10]]
```
Функция **numpy.mean()** применяется для вычисления среднего значения элементов по заданной оси в многомерной матрице:
```
matrix = np.random.randint(0, 10, (9, 9))
a_mean = np.mean(matrix, axis=0)
b_mean = np.mean (matrix,axis=1)
print("Среднее по строкам:", b_mean)
```
Среднее по строкам: [3.88888889 2.88888889 4.88888889 4.55555556 5.11111111 4.22222222 5.33333335. 4.88888889]
```
print("Среднее по столбцам:", a_mean)
```
Среднее по столбцам: [6.22222222 5.11111111 4.55555556 4.55555556 3.77777778 4.22222222 3.66666667 4.22222224.44444444]

9. Транспортирование и изменение формы матриц NumPy. Транспонирование представляет собой процесс замены столбцов на строки и наоборот. Транспонированная матрица отражает результат замены столбцов на строки относительно исходной матрицы. Довольно часто транспонирование используется при работе с векторными произведениями.

Для транспонирования матрицы в библиотеке NumPy применяется функция **np.transpose(A)**. Если массив A имеет размер (n, m), то транспонированный массив будет иметь размер (m, n). Рассмотрим пример транспортирования:
```
import numpy as np
a = np.array([[1, 2, 3],[4, 5, 6]])
print(a)
b = np.transpose(a)
# транспонирование
print(b)
```
В этом примере создается матрица *a* размером 2x3 и выводится на экран. Затем, используя функцию `numpy.transpose()` необходимо выполнить транспонирование этой матрицы и присвоение результата новой

переменной b. На экран выводится новая переменная b, которая является результатом транспонирования.

Исходный массив:

```
[[1 2 3]
 [4 5 6]]
```

Результат:

```
[[1 4]
 [2 5]
 [3 6]]
```

Изменение формы (reshaping). Функция изменения формы позволяет преобразовать матрицу в другую форму без изменения значений элементов. Для этого используется метод **reshape()**.

Допустим, матрицу A можно изменить, чтобы она стала матрицей B с другими размерами и формой. Пусть матрица A имеет форму (n x m), тогда результатом будет матрица B формы (p x q).

Чтобы изменить форму, можно указать новые размеры в виде кортежа в методе reshape:

```
B = A.reshape((p, q)).
```

Рассмотрим пример применения функции reshape():

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12])
arr2 = arr.reshape(4, 3)
print(arr2)
```

```
Вывод: [[ 1  2  3]
        [ 4  5  6]
        [ 7  8  9]
        [10 11 12]]
```

В рассматриваемом примере создается одномерный массив arr1. Затем с помощью функции reshape() он преобразуется в двумерный массив arr2 размером 4x3.

Изменение размеров возможно только при условии неизменности общего количества элементов массива.

3.3 Возможности библиотеки SymPy

Библиотека SymPy - это библиотека символьных вычислений, которая стремится стать полноценной системой компьютерной алгебры. SymPy предоставляет множество функций, которые могут быть полезны в таких

областях, как арифметика, геометрия, алгебра, дискретная математика и квантовая физика. SymPy позволяет форматировать результаты в различных форматах, включая LaTeX и MathML. Библиотека SymPy предоставляет широкий спектр математических возможностей, таких как:

1)Упрощение выражений и раскрытие скобок.

2)Разложение функций в ряд.

3)Вычисление сумм рядов (последовательностей).

4)Вычисление пределов.

5)Вычисление производных функций.

6)Вычисление интегралов.

7)Работа с матрицами (модуль линейной алгебры).

8)Вычисление площадей фигур, образованных пересечением прямых, отрезков и лучей (модуль геометрии).

9)Получение случайных величин с заданной функцией распределения плотности вероятности (модуль статистики).

10) Построение графиков функций и трёхмерных поверхностей, заданных в виде уравнений с символьными переменными.

11) Возможность красивой печати символьных выражений в различных форматах.

Библиотеку SymPy можно использовать в качестве калькулятора. Пример:

```
 from sympy import
a = Rational(1, 2)
b = Rational(1, 3)
print(a, b, sep = ', ')
1/2, 1/3
c = a + b
print(c)
5/6
print(c**2)
25/36
```

С помощью команды `from sympy import` импортируется библиотека SymPy, которая предоставляет возможность работать с математическими выражениями. Далее создаются две переменные: `a` и `b`. Переменной *a* присваивается значение `Rational(1, 2)`, что соответствует рациональному числу `1/2`. Аналогичным образом переменной *b* присваивается значение `1/3`.

Далее с помощью команды `print` выводятся значения переменных `a` и `b`, разделенные запятой - `print(a, b, sep = ' , ')`. В консоли

будет отображено: `1/2, 1/3`, что подтверждает успешное определение значений.

Затем создается новая переменная с именем *c* и ей присваивается сумма переменных, `a` и `b`, которая равна `c = a + b`. После вывода значения переменной *c* на экран командой `print(c)`, в консоли отобразится ответ: `5/6`.

Чтобы возвести `c` в квадрат, необходимо воспользоваться командой `print()` и записать `c**2`. Ответ будет получен в виде дроби: `25/36`.

Для работы с символьными переменными их необходимо правильно описать. Это можно сделать с помощью **функций Symbol()** - для описания одной переменной - и **symbol()** и **var()** - для описания нескольких переменных. Разница между функциями symbol() и var() заключается в том, что последняя добавляет созданные переменные в текущее пространство имён. Рассмотрим пример использования функции Symbol() из модуля SymPy в Python:

```
from sympy import Symbol
x = Symbol('x')
y = x**2
print(y)
```

Вывод:x2**

Этот код определяет символ `x` как переменную в алгебраическом выражении и возводит его в квадрат.

Для раскрытия скобок в символьных выражениях предусмотрена **функция expand**(выражение). Пример использования функции:

```
import sympy
x, y, z = sympy.symbols('x y z')
expression = (x + y) * z
expanded_expression = sympy.expand(expression)
print(expanded_expression)
```

Вывод: x*z + y*z

В данном примере переменные x, y и z объявляются как символы. Создаётся символьное выражение `(x + y)*z`. После этого используется функция `expand()`, которая раскрывает скобки в выражении, и результат сохраняется в `expanded_expression`. В конце выражение выводится на экран.

С помощью функции **subs()** можно подставлять числа в символьные переменные, чтобы получить значение выражений в конкретных точках. Например, необходимо заменить переменные x, y и z на числа везде, где она встречается в выражении. Тогда код будет выглядеть следующим образом:

```
from sympy import symbols
# Создание символьных переменных
x, y, z = symbols('x y z')
# Создание символьного выражения
expr = x + y**2 - z
# Подстановка значений
result = expr.subs({x: 2, y: 3, z: 1})
print(result)
Вывод:10
```

После выполнения этого кода переменная x в выражении expr будет заменена на число 2, переменная y заменена на число 3, переменная z заменена на число 1. В результате выполнения кода ответ будет равен 10.

Функция lambdify() переводит выражения SymPy в функции Python. Если выражение, которое нужно вычислить, затрагивает диапазон значений, то функция evalf() становится неэффективной. В таких случаях lambdify выступает как альтернатива и может использоваться для эффективного вычисления.

Функция действует как лямбда-функция с тем исключением, что она конвертирует SymPy-выражения в имена числовой библиотеки (обычно это NumPy). По умолчанию же она реализована на основе стандартной библиотеки math. Рассмотрим применение функции lambdify() на примере:

```
import sympy as sp
# Создание символьных переменных
x, y = sp.symbols('x y')
# Символьное выражение
expr = x**2 + y*x
# Преобразование символьного выражения (определение функции Python)
func = sp.lambdify(x, expr)
# Вызов функции с заданным аргументом
print(func(5))
# Вывод: 5*y + 25
```

Этот код создаёт символьные переменные x и y, создаёт символьную функцию expr, которая представляет собой $x^2 + xy$, а затем преобразует её в функцию func, которой можно передать любое число вместо x. Затем мы вызываем func с числом 5, что равносильно выражению 5^2+5*y. В результате функция выводит число 5*y + 25.

В библиотеке SymPy предусмотрены функции, которые содержатся в пакете *sympy.plotting*, и позволяют создавать двухмерные и трёхмерные

изображения. Пакет sympy.plotting включает в себя возможность построения следующих функций:

- plot -двухмерные линейные графики.
- plot3d -трехмерные линейные графики.
- plot_parametric -двухмерные параметрические графики.
- plot3d_parametric -трехмерные параметрические графики.

Построение двумерного линейного графика функции рассмотрим на примере графика функции $f(x)=x^2$:

```
from sympy.abc import*
from sympy.plotting import*
plot(x**2, (x, -7, 7))
```

После выполнения команды `plot(x**2, (x, -7, 7))` формируется график функции $f(x)=x^2$ в диапазоне от -7 до 7 (рисунок 11).

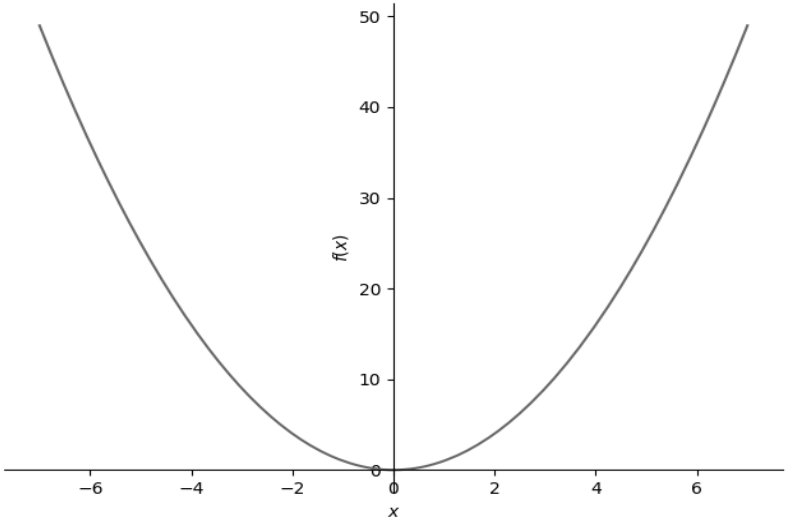

Рисунок 11. Двухмерный линейный график функции $f(x)=x^2$

При построении двумерных графиков функций в пакете sympy.plotting предусмотрена возможность наложения графиков функций. Рассмотрим пример наложения трех функций $f(x)=x$, $f(x)=x^2$ и $f(x)=x^3$:

```
from sympy.abc import*
from sympy.plotting import*
plot(x, x**2, x**3, (x, -5, 5))
```

После выполнения команды `plot(x, x**2, x**3, (x, -5, 5))`

формируется три графика функции в диапазоне от -5 до 5 (рисунок 12).

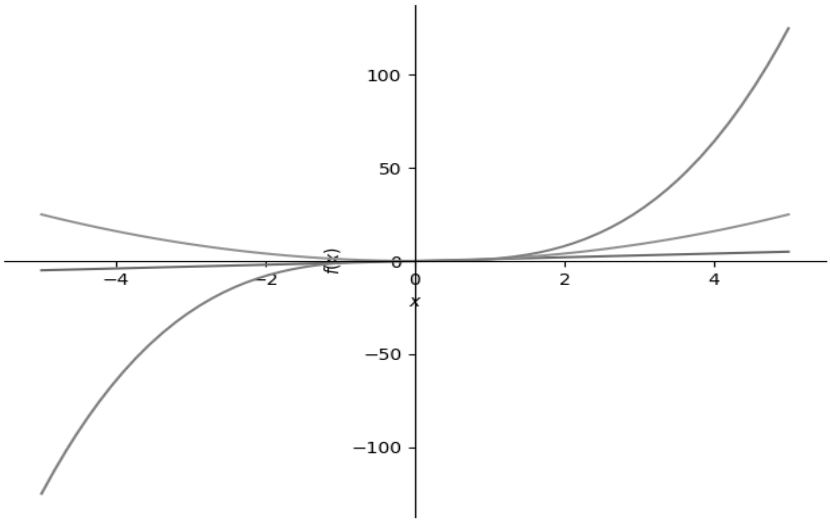

Рисунок 12. Двухмерный линейный график функций $f(x)=x^2$, $f(x)=x^3$, $f(x)=x$

В следующем примере рассмотрена система уравнений: f(x)=x в диапазоне от -5 до 5 и функции $f(x)=x^3$ в диапазоне от -6 до 6 (рисунок 13).

```
from sympy.abc import*
from sympy.plotting import*
plot((x**3, (x, -5, 5)), (x, (x, -6, 6)))
```

Кроме того, при использовании функции plot() можно применять дополнительные аргументы:

- line_color -указывает цвет линии графика;
- title -название графика;
- xlabel -метка для оси X;
- ylabel -метка для оси Y.

Пример построения графика функции с указанием цвета линии и называнием графика представлен на рисунке 14.

```
from sympy.abc import*
from sympy.plotting import*
plot((cos(x),  (x,-2pi,2pi)),  line_color='green',
title='График функции y=cos(x)')
```

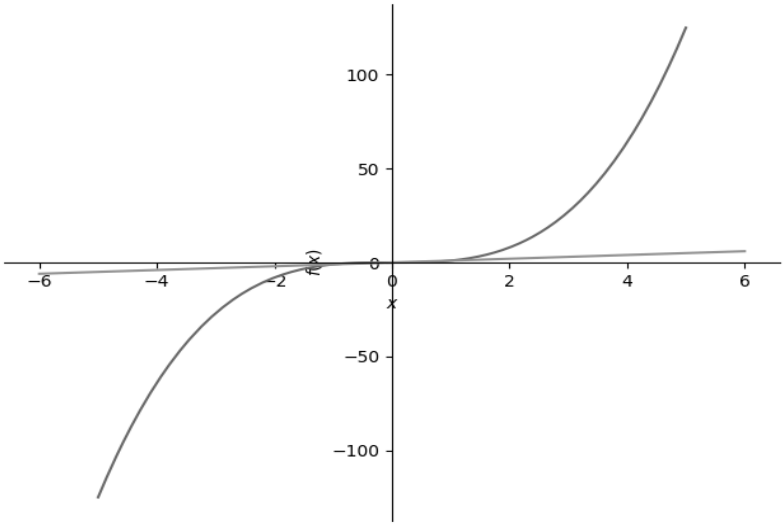

Рисунок 13. Двухмерный линейный график функций $f(x)=x^3$, $f(x)=x$

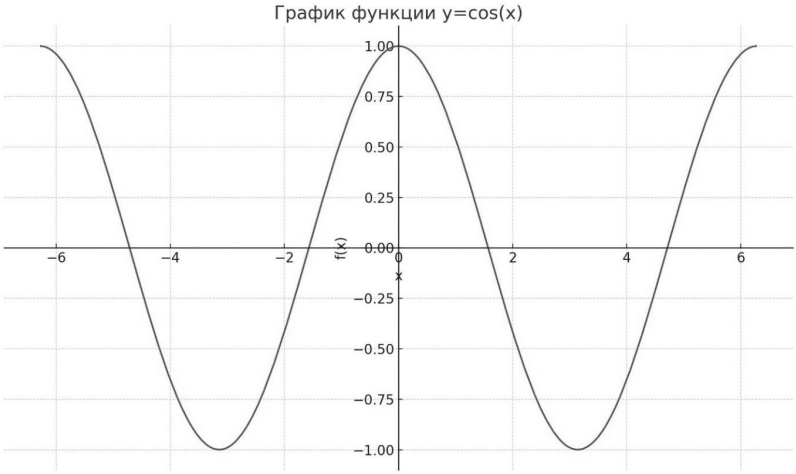

Рисунок 14. Двухмерный линейный график функции y=cos(x)

Функция plot3d. Функция, которая используется для визуализации трёхмерных поверхностей в различных математических пакетах и программах. С её помощью создаются изображения графиков трёхмерных функций, которые помогают представить их форму и поведение.

Для использования plot3d(), требуется задать функцию z = f(x, y), значения границ и шаг изменения переменных x и y. Функция plot3d() даёт наглядное представление о поверхности, что может быть полезно при анализе и исследовании функций, особенно если они имеют сложные графики. Работая с данной функцией можно легко исследовать области максимумов и минимумов, а также выявлять другие особенности функции. Рассмотрим примеры использования функции. В результате выполнения программного кода создаются трехмерные графики функций (рисунки 15, 16, 17).

```
from sympy import*
from sympy.abc import*
from sympy.plotting import*
f = lambda x, y: x**2 + y**2
plot3d(f(x,y))
```

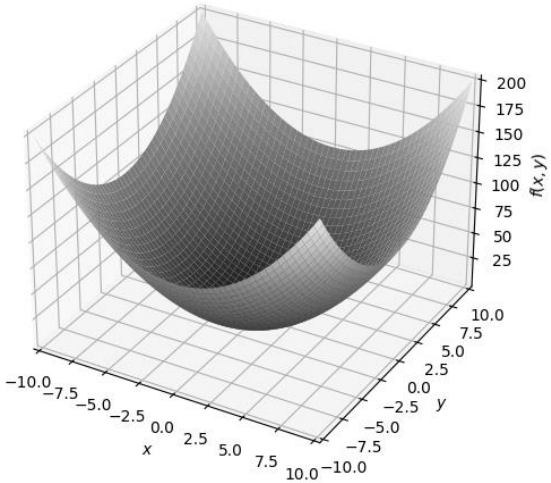

Рисунок 15. Трехмерный линейный график функции $f(x, y) = x^2+y^2$

```
from sympy import*
from sympy.abc import*
from sympy.plotting import*
f = lambda x, y: x**2 + y**2
g = lambda x, y: x * y
plot3d((3*f(x,y), (x, -4, 4), (y, -5, 5)),(g(x,y),
(x, -2, 2), (y, -3, 3)))
```

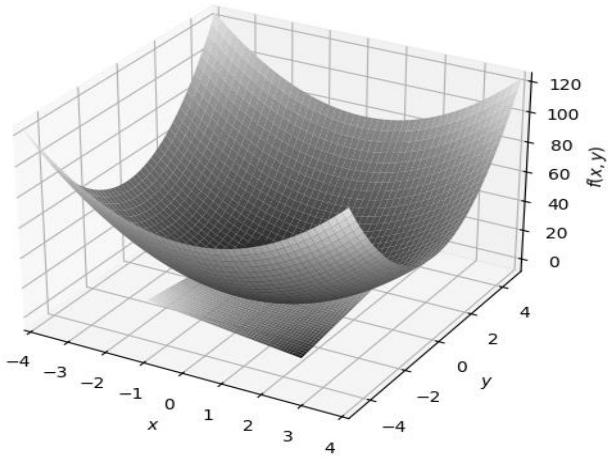

Рисунок 16. Трехмерный линейный график функций 3f(x, y) и g(x, y)

```
from sympy.abc import*
from sympy.plotting import*
f = lambda x, y: x**2 + y**2
plot3d((log(f(x,y)), (x, -5, 5), (y, -5, 5)))
```

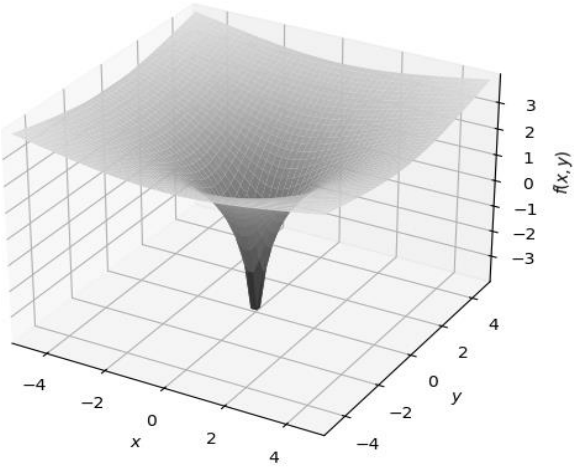

Рисунок 17. Трехмерный линейный график функции log(f(x,y)

Функция plot_parametric. Это функция в SymPy, которая создает параметрический график. Она использует в качестве аргументов две функции f(t) и g(t), которые задают параметрическое уравнение кривой на плоскости, а также переменную t.

Функция plot_parametric вычисляет координаты каждой точки кривой, используя заданные функции и различные значения переменной t между заданными границами. Затем она использует эти точки для построения графика.

Если необходимо, функция plot_parametric преобразует значения кривой в вид, который понимает Matplotlib, и взаимодействует с matplotlib.pyplot для отображения графика. В итоге можно увидеть визуально параметрическую кривую, соответствующую функциям и ее параметрам. Рассмотрим применение функции на примере:

```
import sympy as sp
from sympy.plotting import *
t = sp.symbols('t')
x = sp.sin(2 * t)
y = sp.cos(3 * t) + 1
plot_parametric(x, y, (t, -sp.pi, sp.pi), framework='matplotlib')
```

Результат выполнения программного кода представлен на рисунке 18.

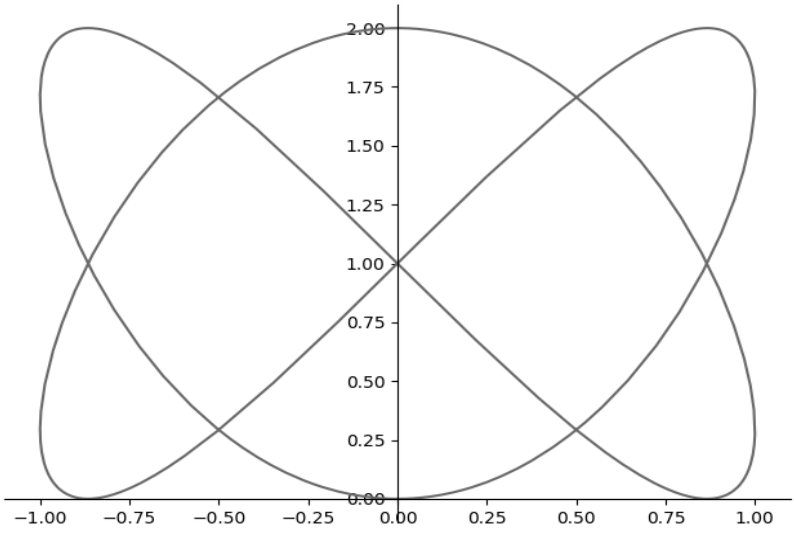

Рисунок 18. Двухмерный параметрический график функций sin(2t), cos(3t) + 1

Функция plot3d_parametric_line. Функция plot3d_parametric_line используется для построения трехмерного параметрического линейного графика. Эта функция может быть частью библиотеки или платформы, предназначенной для создания 3D графиков. Для того, чтобы создать трехмерный график с использованием этой функции, необходимо задать следующие параметры: точки, координаты графика, а также диапазон значений параметров.

Таким образом, пользователь может визуализировать траектории движения точек в трехмерном пространстве путем создания параметрических линейных или других типов трехмерных графиков. Конкретные особенности работы функции зависят от того, в какой программной среде она используется. Например, если эта функция интегрирована в Python, то очевидно, что для ее использования понадобится соответствующий модуль или библиотека. Необходимо импортировать эту функцию, настроить ее параметры и сохранить или отобразить результаты ее работы на графике. Пример использования функции:

```
from sympy.plotting import plot3d_parametric_line
plot3d_parametric_line(sin(x),cos(x), x, (x, -10, 10))
```

Результат выполнения кода представлен на рисунке 19.

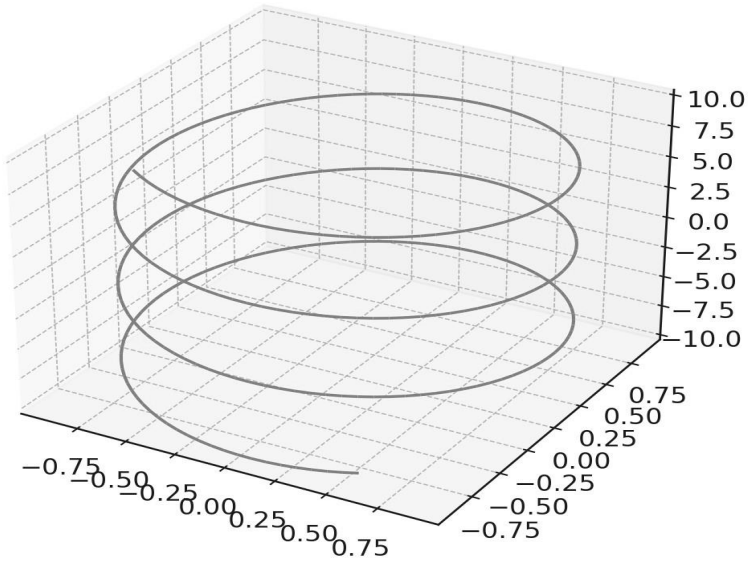

Рисунок 19. Трехмерный параметрический график функций sin(x), cos(x)

3.4 Возможности библиотеки Matplotlib

Библиотека Matplotlib, разработанная для создания двухмерных графиков на основе данных в массиве, является одной из наиболее популярных среди доступных в Python. Matplotlib представляет собой универсальный проект с открытым исходным кодом, который широко используется для визуализации данных и предоставляет возможность создавать графики различных видов. С её помощью можно строить различные типы графиков, такие как линейные графики, трёхмерные графики, диаграммы рассеяния и другие, а также совмещать их между собой.

В библиотеке Matplotlib есть модуль Pyplot, который обычно используется в коде и обозначается *matplotlib.pyplot*. Этот модуль помогает автоматически создавать оси, фигуры и другие компоненты без необходимости разбираться в том, как это устроено.

Вместе с библиотекой Matplotlib устанавливается ещё один модуль - **Pylab**. Он объединяет модуль Pyplot и библиотеку NumPy для работы с массивами. При использовании Pylab можно работать с данными в интерактивном режиме или получать доступ к функциям черчения.

1. **График линейной функции**. График линейной функции является удобным инструментом для наглядного представления и анализа взаимосвязей между двумя переменными. С помощью таких графиков можно проследить динамику изменений данных за определённый период, выявить тенденции и закономерности, а также сравнить значения разных величин. Рассмотрим пример построения линейного графика с помощью библиотеки Matplotlib:

```
from matplotlib import pyplot as plt
   x = [1,2,3,4,5]
y = [10,11,12,13,14]
plt.plot(x,y)
plt.title("Line graph")
plt.ylabel('Y')
plt.xlabel('X')
plt.show()
```

Результат выполнения кода представлен на рисунке 20.

2. Библиотека Matplotlib позволяет создавать графики тригонометрических функций (sin(x), cos(x), tg(x), ctg(x)). Рассмотрим пример создания функции sin(x):

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.arange(0, 5*np.pi, 0.1)
y = np.sin(x)
plt.plot(x, y, color='green')
plt.show()
```

В результате выполнения программы формируется график функции sin(x) (рисунок 21).

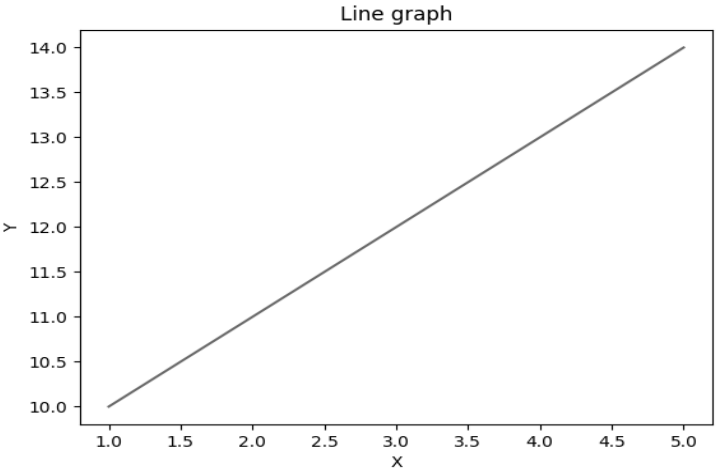

Рисунок 20. Линейный график

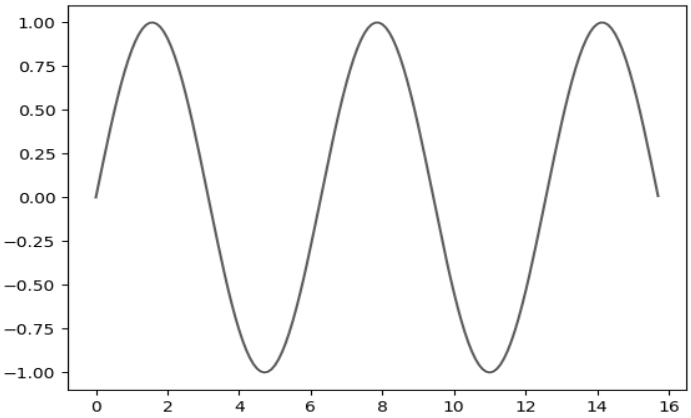

Рисунок 21. График функции sin(x)

3. **Диаграмма рассеяния, или scatterplot** -это тип графика, который используется для анализа взаимосвязи между двумя переменными. Точки на таком графике представляют пары значений двух переменных. Каждая точка соответствует одному наблюдению. Если точки на графике сгруппированы определённым образом, это может указывать на наличие корреляции между переменными. Пример создания диаграммы рассеяния:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [12, 16, 34, 8, 26, 21, 18, 38, 9, 28]
plt.scatter(x, y)
plt.show()
```

В результате выполнения программы диаграмма рассеяния будет выглядеть следующим образом (рисунок 22).

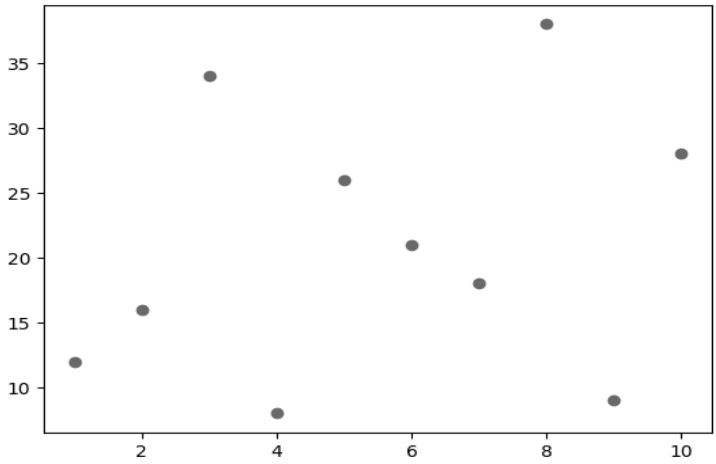

Рисунок 22. Диаграмма рассеяния

4. **Столбчатая диаграмма**. Такой способ визуального представления данных позволяет легко сравнивать значения отдельных переменных. В столбчатой диаграмме длина столбиков соответствует значениям, которые они отображают. Обычно одна из осей диаграммы обозначает категорию, а другая -её дискретное значение.

Например, столбчатая диаграмма помогает визуализировать размер прибыли по месяцам:

```
   import matplotlib.pyplot as plt
   x = ['Январь', 'Февраль', 'Март', 'Апрель', 'Май',
'Июнь', 'Июль']
   y = [5, 2, 8, 4, 7, 6, 1]
   plt.bar(x, y, label='Величина прибыли') #Параметр
label позволяет задать название величины для легенды
   plt.xlabel('Месяц года')
   plt.ylabel('Прибыль, в тыс руб.')
   plt.title('Столбчатая диаграмма')
   plt.legend()
   plt.show()
```

График будет выглядеть следующим образом (рисунок 23).

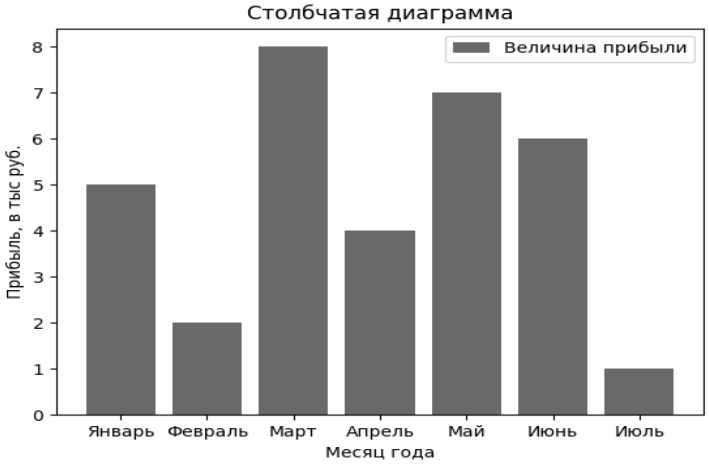

Рисунок 23. Диаграмма рассеяния

5. **Круговая диаграмма**. Круговые диаграммы обычно используются на бизнес-презентациях для наглядного представления информации, такой как продажи, операции, результаты опросов, ресурсы и т. д. Они позволяют быстро и легко считывать информацию и являются хорошим способом представить её в краткой форме. Рассмотрим пример распределения оценок учебной группы за экзамен.

```
   vals = [18, 64, 15, 2, 1]
```

```
   labels = ["Отлично", "Хорошо",
"Удовлетворительно", " Неудовлетворительно ", "Не
аттестован"]
   plt.pie(vals, labels=labels, autopct='%1.1f%%')
   plt.title("Распределение оценок за экзамен")
   plt.show()
```

График будет выглядеть следующим образом (рисунок 24).

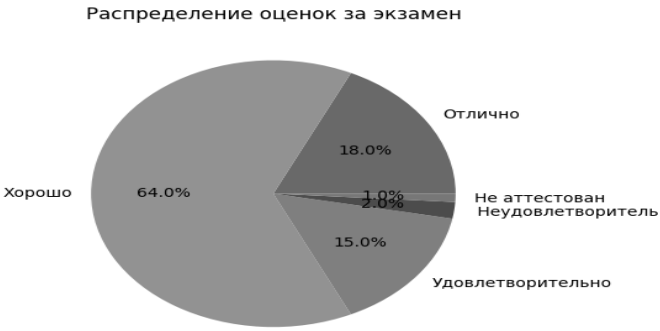

Рисунок 24. Круговая диаграмма

3.5 Возможности библиотеки TensorFlow

Библиотека TensorFlow - это среда машинного обучения с открытым исходным кодом, которая может быть использована для реализации приложений машинного и глубокого обучения. Команда Google создала TensorFlow для разработки и исследования проектов в области искусственного интеллекта. Работа с библиотекой TensorFlow строится на основе построения и выполнения графа вычислений. Граф вычислений - это конструкция, которая описывает то, каким образом будут проводиться вычисления. В классическом императивном программировании пишется код, который выполняется построчно. В TensorFlow привычный императивный подход к программированию необходим только для каких-то вспомогательных целей. Основа TensorFlow - это создание структуры, задающей порядок вычислений. Программы естественным образом структурируются на две части - составление графа вычислений и выполнение вычислений в созданных структурах.

TensorFlow разработан на языке программирования Python, благодаря чему он отличается простотой понимания. TensorFlow позволяет реализовывать и обучать модели машинного обучения с использованием тензоров. Она значительно упрощает построение и тренировку моделей глубокого обучения, делая их более доступными для широкого круга пользователей.

Тензоры - это многомерные массивы или списки, которые используются в качестве базовых структур данных на языке TensorFlow. Они представляют соединяющиеся рёбра на блок-схеме, называемой графом потока данных. Тензором может быть, как отдельное число, вектор признаков из решаемой задачи или изображение, так и целый батч описаний объектов или массив из изображений. Вместо одного объекта можно передать в граф массив объектов и для него будет вычислен массив ответов. Работа TensorFlow с тензорами похожа на то, как обрабатывает массивы NumPy, в функциях которого можно указать ось массива, относительно которой будет выполняться вычисление.

Сессия (session) в TensorFlow - это основной инструмент, который позволяет выполнять операции с тензорами и строить модели машинного обучения. Во время работы с TensorFlow создается граф вычислений (computational graph), который представляет собой набор операций и тензоров. Сессия служит для выполнения этих операций и получения результатов.

Чтобы выполнить вычисления, необходимо создать сессию и запустить её с помощью метода **run()**. Необходимо передать в метод **run()** входные тензоры и описать ожидаемые выходные тензоры.

Когда сессия начинает работать, TensorFlow выполняет операции на графе, используя указанные входные данные. После завершения вычислений результаты становятся доступными в выходных тензорах.

Также сессия может использоваться для отслеживания и управления различными аспектами вычислений. Во время работы сессии можно контролировать параметры обучения, использовать очереди данных и т. д.

Кроме того, сессия в TensorFlow позволяет управлять памятью и ресурсами, необходимыми для вычислений. Возможно контролировать выделение и освобождение памяти, а также следить за использованием других ресурсов. Создадим тензор, заполненный нулями.

```
zeros_tensor = tf.zeros([3, 3])
```

Вообще, API в TensorFlow будет во многом напоминать numpy и **tf.zeros()** - далеко не единственная функция, имеющая прямой аналог в numpy. Чтобы увидеть значение тензора, его нужно выполнить. Подробнее

о выполнении графа чуть ниже, пока что обойдемся тем, что выведем значение тензора и сам тензор.

```
print(zeros_tensor.eval())
print(zeros_tensor)
>>> [[ 0.  0.  0.]
[ 0.  0.  0.]
[ 0.  0.  0.]]
>>>      Tensor("zeros_1:0",      shape=(3,      3),
dtype=float32)
```

Различие между строчками состоит в том, что в первой строке происходит вычисление тензора, а во второй строке просто печатается представление объекта. Описание тензора показывает несколько важных вещей:

1. У тензоров есть имена. У нашего оно `zeros:0`
2. Существует понятие формы тензора, оно похоже на размерность массива из numpy.
3. Тензоры типизированы и типы для них задаются из библиотеки.

Над тензорами можно совершать разнообразные операции. Рассмотрим конструкцию **sess.run()** - это метод исполнения операций графа в сессии.

```
a = tf.truncated_normal([2, 2])
b = tf.fill([2, 2], 0.5)
print(sess.run(a + b))
print(sess.run(a - b))
print(sess.run(a * b))
print(sess.run(tf.matmul(a, b)))
>>> [[-1.12130964 -1.02217746]
[ 0.85684788  0.5425666 ]]
>>> [[ 0.35249496  0.96118248]
[-1.55395389 -1.18111515]]
>>> [[-0.06559008 -0.11100233]
[ 0.51474923 -0.27813852]]
>>> [[-0.16202734 -0.16202734]
[-0.8864761  -0.8864761 ]]
```

В первой строчке тензор из усеченного нормального распределения. Для него используется стандартная генерация нормального распределения, но из него исключается всё, что выпадает за пределы двух стандартных отклонений. Помимо этого, генератора есть равномерное, простое нормальное, гамма и еще несколько других распределений. Второй тензор - это заполненный значением 0.5 многомерный массив размера 2x2.

Далее создаем переменную на основе тензора:

```
v = tf.Variable(zeros_tensor)
```

Переменная участвует в вычислениях в качестве узла вычислительного графа, сохраняет состояние, и ей нужна какая-нибудь инициализация. Так, если в следующем примере обойтись без первой строчки, то TensorFlow выкинет исключение.

```
sess.run(v.initializer)
v.eval()
>>> array([[ 0.,  0.,  0.],
[ 0.,  0.,  0.],
[ 0.,  0.,  0.]], dtype=float32)
```

Операции над переменными создают вычислительный граф, который можно потом выполнить. Еще есть плейсхолдеры - объекты, которые параметризуют граф и отмечают места для подстановки внешних значений. Плейсхолдер - это обещание подставить значение потом. Создадим плейсхолдер и назначаем ему тип данных и размер:

```
x = tf.placeholder(tf.float32, shape=(4, 4))
```

Еще такой пример использования. Два плейсхолдера служат входными узлами для сумматора:

```
a = tf.placeholder("float")
b = tf.placeholder("float")
y = tf.multiply(a, b)
print(sess.run(y, feed_dict={a:100, b:500}))
>>> 50000.0
```

Простейшие вычисления в TensorFlow. В качестве примера создадим и вычислим несколько выражений.

```
x = tf.placeholder(tf.float32)
f =  1 + 2 * x + tf.pow(x, 2)
sess.run(f, feed_dict={x: 10})
>>> 121.0
```

В данном примере, в f мы подставили скалярные значения 1 и 2 и это просто обозначение в графе для чисел (рисунок 25). В этом примере мы создаем плейсхолдер и на его основе - граф выражения 1+2x+x^2, а после этого выполняем вычисления графа в контексте текущей сессии. В примере не указана форма в параметрах плейсхолдера и это значит, что можно подавать на вход тензоры любых размеров. Единственное, что необходимо указать - это тип тензора. Параметры при вычислении внутрь сессии передаются через **feed_dict** - словарь со всем, что необходимо для вычислений.

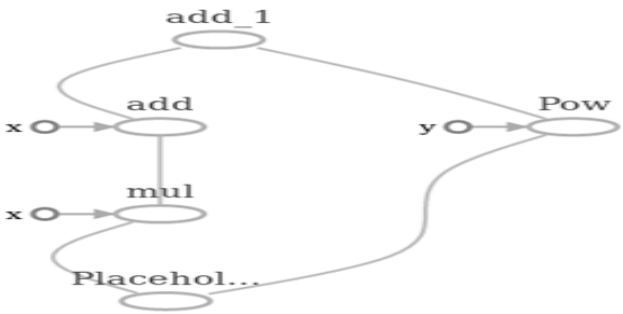

Рисунок 25. Граф вычислений

3.6 Машинное обучение

Искусственный интеллект (ИИ) представляет собой моделирование человеческого интеллекта с помощью машин и компьютерных систем. Он включает в себя такие процессы, как обучение, рассуждение и самокоррекция. Искусственный интеллект способен выполнять множество задач, например, распознавать речь, классифицировать изображения, создавать экспертные системы и даже имитировать человеческое зрение.

Машинное обучение - это область искусственного интеллекта, которая занимается созданием систем и алгоритмов, способных обучаться на основе новых и существующих данных и выявлять в них закономерности.

Чтобы лучше понять концепции машинного и глубокого обучения, можно обратиться к диаграмме Венна (рисунок 26).

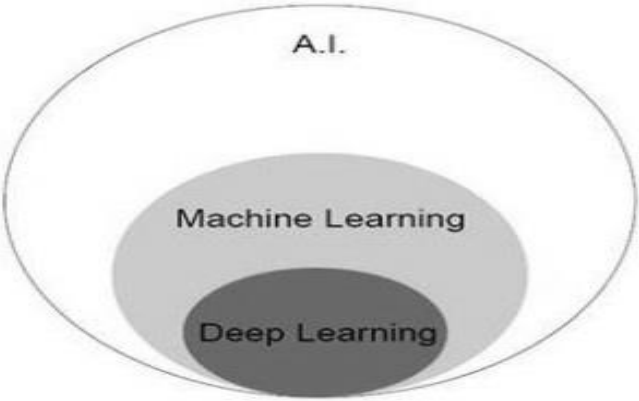

Рисунок 26. Диаграмма Венна

Машинное обучение включает в себя глубокое обучение, которое является его частью. Программы, следующие концепциям машинного обучения, способны улучшать производительность при анализе данных, целью которого является улучшение работы системы в будущем и получение более точных результатов.

Одним из направлений машинного обучения является способность к распознаванию образов -выделению закономерностей в данных и обучению на этих шаблонах для отображения результата определённым образом.

Машинное обучение может выполняться двумя способами: с учителем, когда алгоритм получает размеченные данные и обучается под контролем специалиста, и без учителя, когда система сама находит закономерности в неразмеченных данных. Все задачи, которые решаются с помощью машинного обучения, можно разделить на несколько категорий: классификация, регрессия и кластеризация. Рассмотрим каждую категорию отдельно.

3.6.1 Классификация

Классификация в машинном обучении - это тип задачи обучения с учителем, где модель обучается на данных, для которых известны метки классов, с целью предсказания классов для новых объектов. Это один из основных видов задач, которые решаются с помощью машинного обучения, и она применима в самых разных областях, от распознавания образов и текстов до медицинской диагностики, и финансового прогнозирования. Основные понятия классификации:

1. Метки классов: Данные, используемые для обучения, включают в себя примеры входных данных и соответствующие им метки классов (например, "спам" или "не спам").

2. Бинарная и многоклассовая классификация. Бинарная классификация разделяет данные на два класса (например, определение, больна ли клетка или здорова). Многоклассовая классификация включает в себя три или более классов, между которыми нужно провести различие (например, классификация типов цветов).

3. Алгоритмы классификации. Существует множество алгоритмов для выполнения классификационных задач, каждый со своими особенностями, преимуществами и недостатками. Среди наиболее популярных можно выделить:

Логистическая регрессия. Логистическая регрессия является статистической моделью, используемой для анализа данных, в которых есть одна или несколько независимых переменных, которые определяют исход. Она широко используется для решения задач классификации, таких как прогнозирование вероятности наступления события по данным, имеющим два возможных исхода (например, успех/неудача, да/нет, здоров/болен). Логистическая регрессия предсказывает вероятность принадлежности наблюдения к определенной категории. Отличие от линейной регрессии заключается в том, что зависимая переменная в логистической регрессии является бинарной или дихотомической. Результат моделирования представляет собой вероятность того, что данное наблюдение относится к классу, помеченному как "1", что обычно выражается через функцию логит, связывающую линейную комбинацию входных переменных с логарифмом шансов (отношения вероятностей) для интересующего события. Модель логистической регрессии можно записать следующим образом:

$$log\,\frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots \beta_n x_n\ , \qquad (1)$$

где p -вероятность принадлежности к классу 1,

xi -независимые переменные,

βi -параметры модели, которые необходимо оценить.

Параметры модели обычно оцениваются методом максимального правдоподобия, который ищет такие значения параметров, при которых максимизируется вероятность получения наблюдаемой выборки. Логистическая регрессия находит применение во множестве областей, включая медицину (предсказание вероятности заболеваний), банковское дело (оценка кредитоспособности клиентов), маркетинг (прогнозирование отклика на рекламу), социологические исследования и многое другое. Среди преимуществ логистической регрессии -простота интерпретации результатов, эффективность при небольшом количестве наблюдений и умеренной размерности данных. Недостатки включают предположение о линейной зависимости между независимыми переменными и логарифмом шансов, а также ограничения, налагаемые на распределение данных.

Рассмотрим пример использования логистической регрессии для решения экономической задачи на языке Python. В этом примере используется логистическая регрессия для прогнозирования банкротства компании на основе финансовых показателей. Предположим, есть данные о компаниях, включающие финансовые показатели и информация о том, обанкротилась компания или нет. Мы хотим построить модель, которая будет предсказывать вероятность банкротства компании на основе этих показателей.

Данные могут включать следующие финансовые показатели: общая прибыль, оборотные активы, долгосрочные обязательства, чистая прибыль, коэффициент текущей ликвидности. Целевая переменная: Банкротство (1 - да, 0 - нет).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
# Загрузка данных
data = pd.read_csv('financial_data.csv')
# Предпросмотр данных
print(data.head())
# Определение переменных
X = data[['total_profit', 'current_assets', 'long_term_liabilities', 'net_profit', 'liquidity_ratio']]
y = data['bankruptcy']
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Создание и обучение модели логистической регрессии
model = LogisticRegression()
model.fit(X_train, y_train)
# Предсказания
predictions = model.predict(X_test)
# Оценка модели
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

Этот пример демонстрирует, как можно использовать логистическую регрессию для прогнозирования банкротства компаний на основе их финансовых показателей в Python.

Деревья решений. Деревья решений являются одним из наиболее интуитивно понятных и широко используемых методов классификации в машинном обучении. Они работают путем разделения набора данных на меньшие подмножества на основе информативных признаков, стремясь к созданию подгрупп, которые максимально однородны по целевой переменной. Принципы работы деревьев решений в классификации включает в себя следующие этапы:

Выбор атрибута. На каждом шаге дерево выбирает один атрибут, который наилучшим образом разделяет данные на подгруппы. Этот процесс известен как разделение. Критерии для выбора атрибута могут включать индекс Джини, энтропию и усиленный прирост информации (Information Gain Ratio).

Разделение узла. Когда атрибут выбран, дерево создает ветвь для каждого возможного значения атрибута и разделяет данные в эти ветви.

Рекурсивное разделение. Этот процесс повторяется рекурсивно для каждой ветви, пока не будут выполнены условия остановки. Условия могут включать достижение максимальной глубины дерева, минимального числа образцов в узле или других параметров, которые помогают предотвратить переобучение.

Листовые узлы. В конечном итоге процесс остановится, когда каждый узел будет чистым (т.е. содержит данные только одного класса) или когда будут выполнены другие критерии остановки. Эти узлы называются листьями, и каждому листу присваивается класс на основе большинства классов его образцов.

Рассмотрим пример использования дерева решений для классификации в экономической задаче на языке Python. Задачей будет прогнозирование риска дефолта по кредиту на основе финансовых данных клиентов банка. Предположим, есть набор данных с информацией о клиентах банка, включая их доход, кредитный рейтинг, семейное положение, возраст и другие параметры. Целевая переменная наличия дефолта (1 -дефолт, 0 -нет дефолта).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import tree
import matplotlib.pyplot as plt
# Загрузка данных
data = pd.read_csv('bank_data.csv')
# Предпросмотр данных
print(data.head())
# Определение признаков и целевой переменной
features = ['income', 'credit_rating', 'marital_status', 'age']
X = data[features]
```

```
    y = data['default']
    # Разделение данных на обучающую и тестовую выборки
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    # Создание модели дерева решений
    clf = DecisionTreeClassifier(max_depth=3)
    clf.fit(X_train, y_train)
    # Предсказание на тестовой выборке
    y_pred = clf.predict(X_test)
    # Оценка модели
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    # Визуализация дерева
    plt.figure(figsize=(20,10))
    tree.plot_tree(clf, filled=True, feature_names=features, class_names=['No Default', 'Default'])
    plt.show()
```

Этот пример демонстрирует, как можно использовать дерево решений для решения реальной экономической задачи в области кредитования.

Случайный лес. Случайный лес (*Random Forest*) - это алгоритм машинного обучения, который используется для задач классификации (а также регрессии). Он является ансамблевым методом, основанным на множестве деревьев решений, и одним из самых эффективных и широко применяемых алгоритмов из-за своей высокой точности, устойчивости и легкости интерпретации. Алгоритм состоит из нескольких отдельных деревьев, каждое из которых принимает своё собственное решение относительно класса объекта. Затем все деревья голосуют за класс объекта, и класс с наибольшим количеством голосов становится окончательным решением случайного леса.

Полезная особенность случайного леса в том, что деревья решений обучаются только на случайных подмножествах данных. Также они используют случайное подмножество признаков объекта при построении каждого узла дерева. Этот подход уменьшает вероятность переобучения и повышает общую точность модели.

Улучшение способности к обобщению и снижение вероятности переобучения делают Random Forest одним из наиболее популярных

алгоритмов машинного обучения. Случайные леса успешно применяются в различных областях, таких как распознавание образов, медицинская диагностика, анализ данных о клиентах и многие другие.

Математическая модель для данного набора обучающих данных D с признаками $X=(x_1,x_2,\ldots,x_p)$ и классами $Y=(y_1,y_2,\ldots,y_K)$, случайный лес генерирует N деревьев решений. Каждое дерево Tn обучается на подвыборке Dn, полученной методом бутстрапа. Предсказание класса объекта x случайным лесом RF вычисляется по формуле:

$$\hat{y}=\text{mode}\{T_1(x),T_2(x),\ldots,T_N(x)\}, \quad (2)$$

где mode обозначает наиболее часто встречающийся класс среди предсказаний всех деревьев.

Рассмотрим пример использования алгоритма случайного леса для решения экономической задачи классификации в Python. В качестве примера возьмем задачу кредитного скоринга, где целью является предсказание, сможет ли клиент банка своевременно выплатить кредит на основе его финансовых и личных данных. Предположим, есть данные клиентов банка с такими признаками, как возраст, доход, кредитная история, количество детей и образование. Целевой переменной является факт возврата кредита: 1 (кредит возвращен) или 0 (произошел дефолт).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
# Загрузка данных
data = pd.read_csv('credit_data.csv')
# Закодируем категориальные переменные, если они есть
label_encoders = {}
for column in ['education', 'marital_status']:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le
# Определение признаков и целевой переменной
features = ['age', 'income', 'credit_history', 'children', 'education', 'marital_status']
X = data[features]
y = data['default']
```

```
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
# Создание модели случайного леса
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Предсказания на тестовой выборке
y_pred = model.predict(X_test)
# Оценка модели
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Этот пример демонстрирует, как можно использовать случайный лес для решения задачи кредитного скоринга, обеспечивая высокую точность и устойчивость к переобучению.

Метод опорных векторов (SVM). Метод опорных векторов (*Support Vector Machines, SVM*) - это мощный алгоритм машинного обучения, используемый для задач классификации и регрессии, но наиболее известен своим применением в классификации. SVM стремится найти гиперплоскость (или набор гиперплоскостей) в многомерном пространстве, которая лучше всего разделяет различные классы данных. Основная идея SVM заключается в максимизации зазора между различными классами. Гиперплоскость, которая разделяет классы, должна быть как можно дальше от ближайших точек данных каждого класса, которые называются опорными векторами. В случаях, когда данные не могут быть линейно разделены в исходном пространстве признаков, SVM использует так называемый ядерный трюк. Это позволяет алгоритму оперировать в более высокоразмерном пространстве, где классы могут быть разделены гиперплоскостью. Наиболее популярные ядра включают полиномиальное, радиальное базисное (RBF) и сигмоидное. SVM работает хорошо в случаях, когда количество признаков превышает количество образцов. В самом базовом случае, когда данные линейно разделимы, задача SVM сводится к нахождению гиперплоскости вида:

$$w^T x + b = 0, \quad (3)$$

где w - вектор весов, x - вектор признаков, b - смещение.

Цель SVM - максимизировать зазор между двумя классами, что эквивалентно минимизации нормы вектора весов $\|w\|$. Проблема оптимизации формулируется как:

$$\min_{w,b} \frac{1}{2} \| w \|^2 \quad (4)$$

при условии $y_i(w^T x_i + b) \geq 1$, $i=1,\ldots,n$

где y_i -метки классов, которые принимают значения −1 или 1, x_i - обучающие образцы, n -количество обучающих образцов.

Когда данные не линейно разделимы, SVM использует так называемый ядерный трюк для проекции данных в более высокоразмерное пространство, где линейное разделение становится возможным. Ядро $K(x,x')$ представляет собой функцию, которая соответствует скалярному произведению двух векторов в пространстве с более высокой размерностью. Задача оптимизации с использованием ядерной функции изменяется следующим образом:

$$\min_{\alpha} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i x_j) - \sum_i \alpha_i, \quad (5)$$

при условии $0 \leq \alpha_i \leq C$, $\sum_i a_i y_i = 0$

где α_i -переменные Лагранжа,

C -параметр регуляризации, который контролирует компромисс между максимизацией зазора и минимизацией количества ошибочных классификаций.

Решение этой задачи дает набор коэффициентов α_i, большинство из которых будет равно нулю. Ненулевые α_i соответствуют опорным векторам, которые непосредственно влияют на форму гиперплоскости.

Рассмотрим пример использования метода опорных векторов (SVM) для решения экономической задачи классификации в Python. В качестве задачи выберем классификацию клиентов банка на тех, кто вероятно выплатит кредит, и тех, кто может допустить дефолт, на основе их финансовых данных. Предположим, есть данные о клиентах банка, включая такие признаки, как возраст, доход, количество кредитных карт, наличие долгов и другие. Целевая переменная -это факт возврата кредита: 1 (кредит возвращен) или 0 (произошел дефолт).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

```
from sklearn.metrics import classification_report,
confusion_matrix, accuracy_score
# Загрузка данных
data = pd.read_csv('bank_customers.csv')
# Предпросмотр данных
print(data.head())
# Определение признаков и целевой переменной
features = ['age', 'income', 'credit_cards',
'debts']
X = data[features]
y = data['default']
# Нормализация данных
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y, test_size=0.3,
random_state=42)
# Создание модели SVM
model = SVC(kernel='linear') # Линейное ядро
используется для примера
model.fit(X_train, y_train)
# Предсказание на тестовой выборке
y_pred = model.predict(X_test)
# Оценка модели
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n",
confusion_matrix(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
```

Этот пример демонстрирует, как можно использовать SVM для решения задачи классификации в экономической сфере, обеспечивая высокую точность и эффективность обработки данных.

k-ближайших соседей (k-NN). Метод K-ближайших соседей является простым и популярным алгоритмом машинного обучения, основанный на предположении, что схожие объекты находятся рядом друг с другом. В основе алгоритма лежит понятие окрестности или k-окрестности. Для классификации объекта алгоритм находит k-объектов из обучающей выборки, которые находятся ближе всего к объекту, и определяет наиболее часто встречаемый класс среди k -соседей как класс нового объекта. То есть

классифицируемый объект присваивается к классу, который является наиболее распространённым среди его K-ближайших «соседей», или объектов из обучающего множества. Поэтому значение имеет метрика расстояния между объектами. Выбор значения параметра k является важным шагом в алгоритме k -NN, потому что он сильно влияет на точность классификации и скорость работы алгоритма. Высокое значение K означает, что для определения нового объекта будет учитываться большее количество соседей, но это может привести к уменьшению точности классификации. Низкое значение k может увеличить точность, но может стать причиной переобучения модели и снизить обобщающую способность модели. Это связано с тем, что небольшое количество соседних объектов может не представлять весь спектр данных, и модель будет слишком зависеть от них.

Рассмотрим пример использования алгоритма k-ближайших соседей (k-NN) для решения экономической задачи классификации в Python. Возьмем в качестве задачи прогнозирование надежности кредитных заемщиков на основе их финансовой информации. Предположим, есть данные о заемщиках, включая такие признаки как возраст, годовой доход, количество лет на текущем месте работы, сумма кредита и история предыдущих кредитов. Целевая переменная -это факт своевременного возврата кредита: 1 (кредит возвращен) или 0 (произошел дефолт).

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
# Загрузка данных
data = pd.read_csv('loan_data.csv')
# Предпросмотр данных
print(data.head())
# Определение признаков и целевой переменной
features = ['age', 'annual_income', 'years_at_job', 'loan_amount', 'credit_history']
X = data[features]
y = data['loan_returned']
# Нормализация данных
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Разделение данных на обучающую и тестовую выборки
```

```
   X_train,      X_test,      y_train,      y_test      =
train_test_split(X_scaled,      y,      test_size=0.25,
random_state=42)
   # Создание модели k-NN
   knn = KNeighborsClassifier(n_neighbors=5)
   knn.fit(X_train, y_train)
   # Предсказание на тестовой выборке
   y_pred = knn.predict(X_test)
   # Оценка модели
   print("Accuracy:", accuracy_score(y_test, y_pred))
   print("Confusion                          Matrix:\n",
confusion_matrix(y_test, y_pred))
   print("Classification Report:\n",
```

3.6.2 Регрессия

В методах машинного обучения существуют различные виды регрессий:

1. **Линейная регрессия** - это метод, который пытается предсказать непрерывную выходную переменную на основе входных данных, используя линейную функцию. В линейной регрессии отношения между переменными моделируются с использованием линейных предикторных функций, и неизвестные параметры модели (веса) оцениваются на основе данных. Существуют два основных типа линейной регрессии:

Простая линейная регрессия. Используется, когда зависимая переменная (целевая) и одна независимая переменная (предиктор) связаны линейной зависимостью. Формула для простой линейной регрессии:

$$y = \beta_0 + \beta_1 x + \epsilon, \quad (6)$$

где y - зависимая переменная, x - независимая переменная, β_0 и β_1 - параметры модели, которые нужно оценить, ϵ - ошибка модели, предполагается, что она нормально распределена.

Множественная линейная регрессия. Применяется, когда зависимая переменная должна быть предсказана на основе нескольких независимых переменных. В множественной линейной регрессии уравнение становится более сложным, включая несколько независимых переменных:

$$y=\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta n xn+\epsilon \quad (7)$$

Наиболее распространенным методом для оценки параметров (β) является метод наименьших квадратов, который минимизирует сумму квадратов разностей между наблюдаемыми значениями и значениями,

предсказанными моделью. Этот метод может быть реализован аналитически через нормальное уравнение или численно с помощью различных оптимизационных алгоритмов, таких как градиентный спуск.

Линейная регрессия используется для прогнозирования значений, анализа трендов и определения взаимосвязей между переменными. Однако метод имеет ограничения:

- Предполагает линейную зависимость между переменными.
- Чувствителен к выбросам в данных.
- Может страдать от мультиколлинеарности, когда независимые переменные сильно коррелированы.

Рассмотрим пример линейной регрессии в Python, решающий экономическую задачу - прогнозирование ВВП на основе инвестиций в технологии. Необходимо построить модель линейной регрессии для одной независимой переменной и визуализируем результаты с помощью графика (рисунок 27). Предположим, что есть данные за несколько лет по инвестициям в технологии и росту ВВП (таблица 5).

Таблица 5 - Исходные данные

Год	Инвестиции в технологии (млн USD)	ВВП (млрд USD)
2019	200	1800
2020	220	1850
2021	250	1950
2022	270	2100
2023	300	2300

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
# Данные
X = np.array([[200], [220], [250], [270], [300]])
# Инвестиции в технологии
y = np.array([1800, 1850, 1950, 2100, 2300])  # ВВП
# Создание и обучение модели линейной регрессии
model = LinearRegression()
model.fit(X, y)
# Визуализация данных и линии регрессии
plt.scatter(X, y, color='blue', label='Исходные данные')  # Рассеянные точки данных
plt.plot(X, model.predict(X), color='red', label='Линия линейной регрессии')  # Линия регрессии
plt.xlabel('Инвестиции в технологии (млн USD)')
```

```
    plt.ylabel('ВВП (млрд USD)')
    plt.title('Линейная регрессия для прогнозирования ВВП')
    plt.legend()
    plt.show()
```

Этот код предоставляет базовый пример построения линейной регрессии для экономической задачи, делая акцент на возможности предсказания будущих экономических показателей на основе инвестиционных данных.

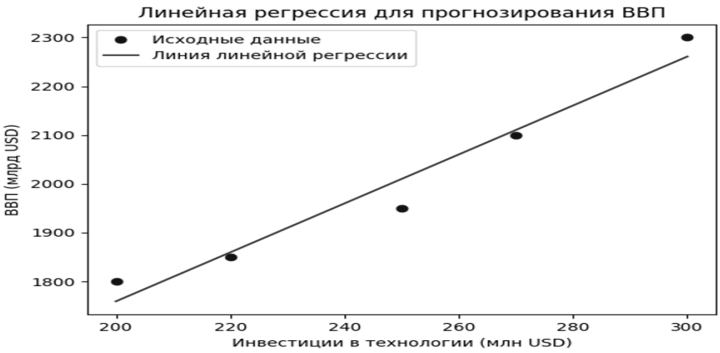

Рисунок 27. График линейной регрессии

2. **Ридж-регрессия**, также известная как гребневая регрессия, представляет собой метод регуляризации линейной регрессии. Он используется для уменьшения переобучения и обеспечения стабильности модели. Основная цель ридж-регрессии - уменьшить сложность модели и предотвратить переобучение, что часто происходит при использовании стандартной модели линейной регрессии. Основные аспекты ридж-регрессии:

1) **Штраф за величину коэффициентов**. Ридж-регрессия минимизирует сумму квадратов ошибок, как и обычная линейная регрессия, но также добавляет штраф за размер коэффициентов. Этот штраф -сумма квадратов коэффициентов, умноженных на параметр регуляризации λ. Формула для функции потерь ридж-регрессии:

$$L(\beta) = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \quad (8)$$

где *yi* -наблюдаемые значения, *xi* -вектор предикторов, β -вектор коэффициентов, λ -параметр регуляризации.

2) **Выбор параметра λ**: Величина λ определяет степень влияния штрафа на общую функцию потерь. При λ=0 ридж-регрессия сводится к обычной линейной регрессии. При очень больших значениях λ коэффициенты могут стремиться к нулю, что приводит к подавлению некоторых переменных в модели.

3) **Решение уравнения ридж-регрессии**. Оптимальные значения коэффициентов β могут быть найдены аналитически через:

$$\beta = (X^T X + \lambda I)^{-1} X^T y \tag{9}$$

где X - матрица наблюдений, I - единичная матрица, а размерность соответствует числу предикторов.

Рассмотрим пример ридж-регрессии в Python, решающий экономическую задачу прогнозирования ВВП на основе двух переменных: инвестиций в образование и уровня безработицы. Сформируем набор данных:

```
import numpy as np
# Инвестиции в образование (млн USD) и уровень безработицы (%)
X = np.array([
    [500, 5],
    [600, 4.8],
    [700, 4.6],
    [800, 4.4],
    [900, 4.2]
])
# ВВП (млрд USD)
y = np.array([1800, 1900, 2000, 2100, 2200])
```

Затем используем библиотеку scikit-learn для построения ридж-регрессии и визуализируем ее на графике функции (рисунок 28).

```
from sklearn.linear_model import Ridge
import matplotlib.pyplot as plt
# Параметры регуляризации
alphas = np.logspace(-6, 6, 200)
# Сохранение коэффициентов
coefs = []
for a in alphas:
    ridge = Ridge(alpha=a)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)
# Визуализация изменения коэффициентов
```

```
    plt.figure(figsize=(10, 6))
    ax = plt.gca()
    ax.plot(alphas, coefs)
    ax.set_xscale('log')
    plt.xlabel('Параметр регуляризации (alpha)')
    plt.ylabel('Коэффициенты')
    plt.title('Влияние регуляризации на коэффициенты
ридж-регрессии')
    plt.legend(['Инвестиции в образование', 'Уровень
безработицы'])
    plt.show()
```

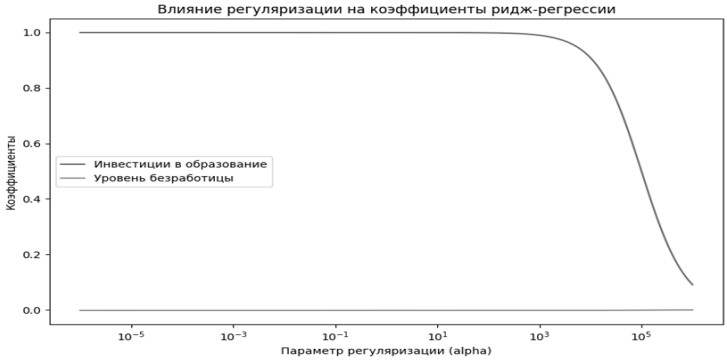

Рисунок 28. График ридж-регрессии

3. **ЛАССО регрессия, или метод наименьших квадратов с лассо-ограничением**. ЛАССО уменьшает размерность модели путём исключения некоторых предикторов. Основные характеристики лассо регрессии:

• L1-регуляризация. В лассо регрессии штраф добавляется к абсолютным значениям коэффициентов модели. Целевая функция лассо один тип линейной регрессионной модели, который используется для предотвращения переобучения регрессии имеет вид:

$$L(\beta) = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j| \qquad (10)$$

где *yi* -наблюдаемые значения, *xi* -вектор предикторов, β -вектор коэффициентов, λ -параметр регуляризации, контролирующий степень сжатия коэффициентов.

• Выбор переменных. Одно из ключевых преимуществ лассо регрессии -её способность обнулять коэффициенты менее важных переменных. Это делает лассо регрессию особенно полезной, когда предполагается, что многие признаки не влияют на зависимую переменную, облегчая интерпретацию модели и уменьшая размерность данных.

• Борьба с переобучением. Подобно ридж-регрессии, лассо регрессия также помогает предотвращать переобучение за счет уменьшения значений коэффициентов.

• Вычислительные аспекты. Хотя решение для лассо регрессии не может быть выражено в замкнутой форме (как в ридж-регрессии), оно эффективно находится с помощью численных методов оптимизации, таких как координатный спуск.

Рассмотрим пример ЛАССО регрессии в Python, применяя его для экономической задачи прогнозирования дохода населения на основе данных о расходах на образование и уровне безработицы. Предположим, есть следующий набор данных, содержащий данные за несколько лет (таблица 6).

Таблица 6 - Исходные данные

Год	Расходы на образование (млн USD)	Уровень безработицы (%)	Доход на душу населения (USD)
2019	300	5.5	28000
2020	320	5.3	29000
2021	340	5.1	30000
2022	360	4.9	31000
2023	380	4.7	32000

Для построения модели лассо регрессии используется библиотека scikit-learn:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
# Данные
X = np.array([[300, 5.5], [320, 5.3], [340, 5.1], [360, 4.9], [380, 4.7]])  # Расходы и безработица
y = np.array([28000, 29000, 30000, 31000, 32000])  # Доход
# Создание и обучение модели лассо регрессии с параметром регуляризации alpha
alpha = 0.1  # Начальное значение alpha
lasso_model = Lasso(alpha=alpha)
lasso_model.fit(X, y)
```

```
    # Вывод коэффициентов
    print("Коэффициенты модели:", lasso_model.coef_)
    print("Свободный член:", lasso_model.intercept_)
    # Предсказание на новых данных
    new_data = np.array([[400, 4.5]])
    predicted_income = lasso_model.predict(new_data)
    print("Прогнозируемый доход на душу населения:",
predicted_income[0])
    # Визуализация изменения коэффициентов при разных
значениях alpha
    alphas = np.logspace(-4, 1, 30)
    coefs = []
    for a in alphas:
        lasso = Lasso(alpha=a)
        lasso.fit(X, y)
        coefs.append(lasso.coef_)
    plt.figure(figsize=(10, 6))
    plt.plot(alphas, coefs)
    plt.xscale('log')
    plt.xlabel('Параметр регуляризации (alpha)')
    plt.ylabel('Коэффициенты')
    plt.title('Влияние регуляризации на коэффициенты
лассо регрессии')
    plt.legend(['Расходы на образование', 'Уровень
безработицы'])
    plt.show()
```

Визуализация ЛАССО регрессии представлена на рисунке 29.

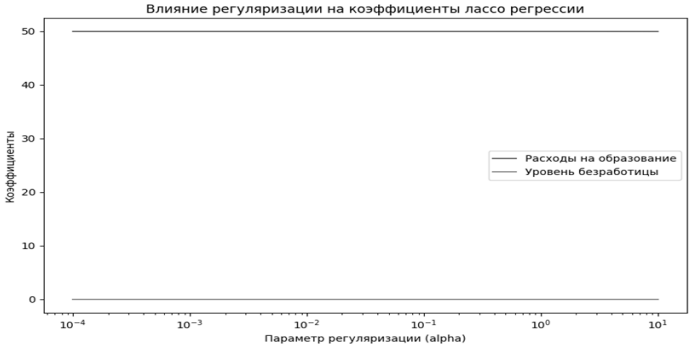

Рисунок 29. График ЛАССО регрессии

4. **Полиномиальная регрессия** представляет из себя метод множественной регрессии, использующий полиномы (переменные степени) в качестве предикторов для предсказания непрерывной выходной переменной. Модель регрессии представлена формулой:

$$y=\beta_0+\beta_1 x+\beta_2 x2+\ldots+\beta_n xn+\epsilon, \quad (11)$$

где $\beta_0,\beta_1,\ldots,\beta_n$ -параметры модели, ϵ -ошибка модели

Рассмотрим пример полиномиальной регрессии в Python, где мы попробуем предсказать ВВП страны на основе уровня инвестиций. Полиномиальная регрессия может быть особенно полезной, когда взаимосвязь между независимой и зависимой переменной не является линейной, но можно описать более высоким порядком полинома. Предположим, есть данные об инвестициях и соответствующем ВВП за несколько лет (таблица 7).

Таблица 7 - Исходные данные

Год	Инвестиции в технологии (млн USD)	ВВП (млрд USD)
2019	250	1800
2020	300	1850
2021	350	1950
2022	270	2100
2023	450	2300

Используем библиотеки NumPy для работы с данными и Scikit-learn для построения модели полиномиальной регрессии.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
# Данные
X = np.array([[250], [300], [350], [400], [450]])
# Инвестиции
y = np.array([1800, 1850, 1950, 2100, 2300])
# ВВП
# Преобразование данных в полиномиальные признаки
poly = PolynomialFeatures(degree=2)  # Используем полином второй степени
X_poly = poly.fit_transform(X)
# Создание и обучение модели линейной регрессии на преобразованных данных
model = LinearRegression()
model.fit(X_poly, y)
```

```
    # Визуализация
    plt.scatter(X, y, color='red', label='Исходные данные')
    # Генерация данных для линии предсказаний
    X_fit = np.linspace(250, 450, 100).reshape(-1, 1)
    X_fit_poly = poly.transform(X_fit)
    y_fit = model.predict(X_fit_poly)
    plt.plot(X_fit, y_fit, label='Полиномиальная регрессия')
    plt.xlabel('Инвестиции (млн USD)')
    plt.ylabel('ВВП (млрд USD)')
    plt.title('Полиномиальная регрессия для прогнозирования ВВП')
    plt.legend()
    plt.show()
```

На графике (рисунок 30) представлена визуализация результатов полиномиальной регрессии для прогнозирования ВВП на основе уровня инвестиций. Красные точки обозначают исходные данные, а синяя линия показывает предсказания модели, использующей полином второй степени. Эта линия наглядно демонстрирует нелинейную зависимость между инвестициями и ВВП, которую полиномиальная регрессия способна улавливать и моделировать.

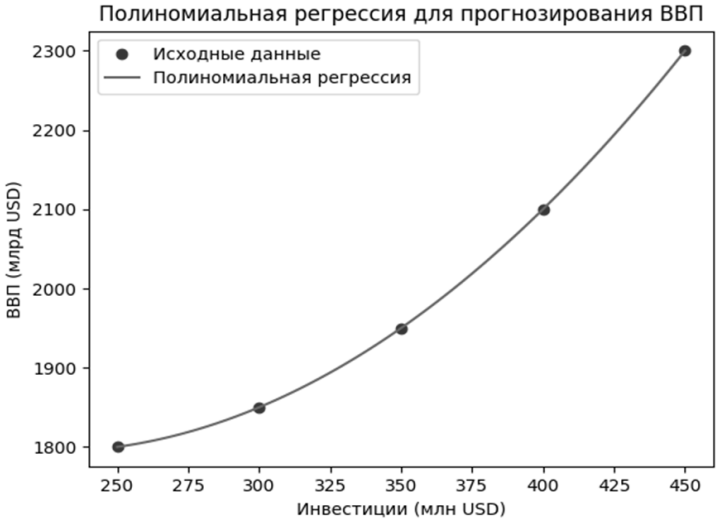

Рисунок 30. График полиномиальной регрессии

5. **Регрессия дерева решений** — это алгоритм машинного обучения, основанный на деревьях решений, где каждый внутренний узел дерева представляет условие, а каждое внешнее разветвление представляет возможное решение или ответ.

Каждый узел дерева представляет вопрос или условие, разделяющее данные на два подмножества. Листья дерева представляют прогнозируемые значения зависимой переменной, которые обычно являются средним значением целевой переменной для наблюдений в этом листе.

Рассмотрим пример использования регрессии дерева решений в Python, на этот раз для прогнозирования уровня инфляции на основе данных о государственных расходах и безработице. Это позволит понять, как изменения в экономических индикаторах могут влиять на инфляцию.

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
# Генерация искусственных данных
np.random.seed(0)
n_samples = 100
government_spending = 100 + 15 * np.random.randn(n_samples)
unemployment_rate = 5 + 2 * np.random.randn(n_samples)
inflation_rate = 0.5 * government_spending - 0.3 * unemployment_rate + 10 + 5 * np.random.randn(n_samples)
# Подготовка данных для модели
X = np.column_stack((government_spending, unemployment_rate))
y = inflation_rate
# Создание и обучение модели
model = DecisionTreeRegressor(max_depth=4)
model.fit(X, y)
# Генерация сетки для визуализации
xx, yy = np.meshgrid(np.linspace(80, 130, 500),
                     np.linspace(1, 9, 500))
Z = model.predict(np.c_[xx.ravel(),yy.ravel()]).reshape(xx.shape)
# Визуализация
plt.figure(figsize=(10, 7))
plt.contourf(xx, yy, Z, alpha=0.3)
```

```
    plt.scatter(government_spending, unemployment_rate,
c=inflation_rate, cmap='Reds')
    plt.colorbar(label='Инфляция (%)')
    plt.xlabel('Государственные расходы')
    plt.ylabel('Уровень безработицы')
    plt.title('Прогноз уровня инфляции на основе
государственных расходов и уровня безработицы')
    plt.show()
```

На представленном графике (рисунок 31) визуализирована модель регрессии дерева решений, которая прогнозирует уровень инфляции на основе государственных расходов и уровня безработицы. Контурная карта показывает предсказания модели, где различные уровни инфляции отображаются разными цветами. Точки на графике представляют исходные данные, цвет которых соответствует фактическому уровню инфляции. Это визуализация помогает увидеть, как изменение государственных расходов и уровня безработицы влияет на уровень инфляции в экономике.

Этот пример иллюстрирует мощь деревьев решений в моделировании сложных экономических взаимосвязей и предоставляет наглядное представление о влиянии различных экономических факторов на уровень инфляции.

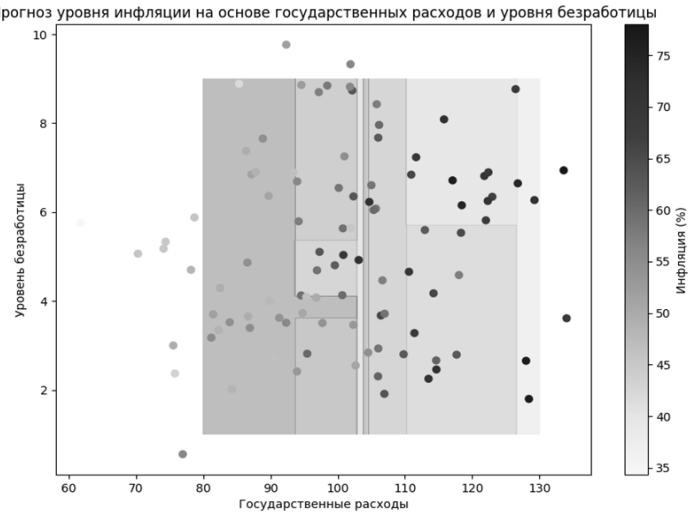

Рисунок 31. График регрессии дерева решений

5. **Случайный лес регрессии** использует множество деревьев решений и усредняет их предсказания для повышения точности модели. Случайные подмножества признаков используются для создания отдельных деревьев. Такая генерация деревьев решений помогает снизить возможность переобучения модели. Метод может использоваться в случаях большого объёма данных. Основные характеристики случайного леса в регрессии:

- Ансамблевая модель. Случайный лес строит множество деревьев решений, каждое из которых обучается на разных подвыборках данных (с использованием метода бутстраппинга) и на подмножестве доступных признаков, что улучшает обобщающую способность модели и уменьшает риск переобучения.
- Устойчивость к переобучению. Благодаря использованию множества деревьев и случайности в выборе признаков для каждого дерева, случайный лес обычно демонстрирует лучшую устойчивость к переобучению по сравнению с одним деревом решений.
- Важность признаков. Случайный лес автоматически оценивает важность каждого признака в процессе обучения, что делает его полезным для выбора признаков в больших и сложных наборах данных.
- Гиперпараметры. Основные параметры, которые нужно настроить в случайном лесе, включают число деревьев в лесу, глубину деревьев, минимальное число образцов, необходимое для разделения узла, и минимальное число образцов для листа.

Рассмотрим пример модели случайного леса для регрессии, в котором прогнозируются цены на жильё в Москве, используя искусственные данные о характеристиках квартир. Параметры будут включать площадь квартиры, расстояние до центра города, количество комнат и год постройки дома. Данные:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
# Генерация искусственных данных
np.random.seed(42)
n_samples = 300
area = np.random.rand(n_samples) * 120 + 30  # площадь от 30 до 150 кв. м
```

```
    distance_to_center = np.random.rand(n_samples) * 15
# расстояние до центра в км
    rooms = np.random.randint(1, 5, size=n_samples)  #
количество комнат от 1 до 4
    year_built    =    np.random.randint(1950,    2021,
size=n_samples)  # год постройки
    # Симуляция цены в зависимости от характеристик
    prices = (area * 2000 + distance_to_center * (-1500)
+ rooms * 50000 + (2020 - year_built) * -30
              + np.random.randn(n_samples) * 50000)
    # Подготовка данных для модели
    X   =   np.column_stack((area,   distance_to_center,
rooms, year_built))
    y = prices
    # Разбиение данных на обучающую и тестовую выборки
    X_train,     X_test,     y_train,     y_test     =
train_test_split(X, y, test_size=0.2, random_state=42)

    # Создание модели случайного леса
    model   =   RandomForestRegressor(n_estimators=100,
max_depth=10, random_state=42)
    model.fit(X_train, y_train)
    # Предсказание и визуализация
    y_pred = model.predict(X_test)
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, color='red')
    plt.plot([y_test.min(),              y_test.max()],
[y_test.min(), y_test.max()], 'k--', lw=4)
    plt.xlabel('Фактические цены')
    plt.ylabel('Предсказанные цены')
    plt.title('Фактические против предсказанных цен на
жильё в Москве')
    plt.show()
```

На графике выше (рисунок 32) показано сравнение фактических и предсказанных цен на жильё в Москве, полученных с помощью модели случайного леса. Точки красного цвета представляют собой пары фактических и предсказанных цен из тестовой выборки. Черная пунктирная линия показывает идеальное соответствие, где предсказанные цены равны фактическим.

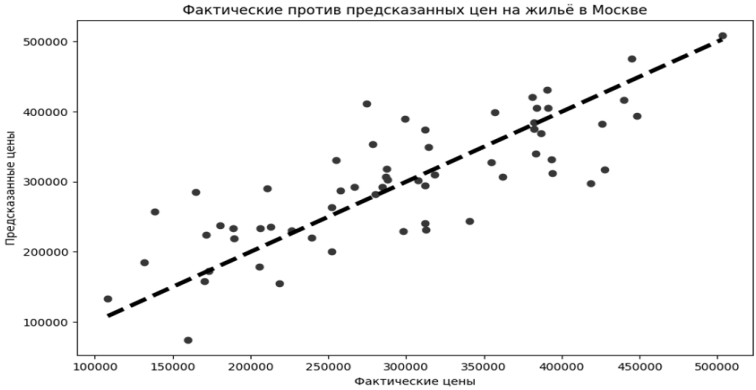

Рисунок 32. График регрессии случайного леса

Эта визуализация демонстрирует, насколько близко предсказания модели соответствуют реальным данным. Большинство точек находится вблизи идеальной линии, что указывает на высокую точность предсказаний модели случайного леса для данной задачи. Это подчеркивает способность случайного леса обрабатывать сложные зависимости и предоставлять точные прогнозы в экономических приложениях, таких как оценка стоимости недвижимости.

7. **Опорная векторная регрессия** (SVR) - это вспомогательная функция, которая помогает строить модель машинного обучения. SVR находит гиперплоскость, которая лучше всего разделяет данные, для точного прогнозирования. Этот метод также используется для решения задач регрессии и предназначен для минимизации ошибок. Основные концепции SVR:

- Эпсилон-трубка (epsilon-tube). SVR стремится найти функцию, значения которой для всех обучающих данных попадают в epsilon-окрестность от истинных значений, минимизируя при этом общую сложность модели.
- Функция потерь. Основной особенностью SVR является использование линейной функции потерь, называемой функцией потерь "epsilon-insensitive", которая не учитывает ошибки, находящиеся в пределах epsilon от истинного значения. Это означает, что ошибки, меньшие заданной величины epsilon, в модели не учитываются как таковые.
- Ядра. Подобно SVM для классификации, SVR может использовать различные ядра (линейное, полиномиальное, радиально-базисное и др.) для

работы в пространствах высокой размерности, что позволяет аппроксимировать нелинейные зависимости.

• Регуляризация. Через параметр регуляризации C модели можно контролировать компромисс между снижением эмпирического риска (точность аппроксимации) и уменьшением сложности модели (гладкость функции).

Рассмотрим пример использования опорной векторной регрессии (SVR) для анализа цен на жильё в Москве на основе данных, таких как площадь квартиры, расстояние до центра города и этаж.

```
import numpy as np
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Искусственные данные
np.random.seed(42)
n_samples = 200
area = np.random.rand(n_samples) * 120  # площадь от 20 до 140 кв. м
distance_to_center = np.random.rand(n_samples) * 20  # расстояние до центра в км
floor = np.random.randint(1, 30, size=n_samples)  # этаж
# Цена квартиры зависит от этих факторов + случайный шум
prices = 150000 + area * 60000 + (30 - distance_to_center) * 1500 + floor * 2000 + np.random.randn(n_samples) * 50000
# Подготовка данных для SVR
X = np.column_stack((area, distance_to_center, floor))
y = prices
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Создание модели SVR
svr_model = SVR(kernel='rbf', C=10000, epsilon=5000)
svr_model.fit(X_train, y_train)
# Предсказание цен на тестовых данных
```

```
    y_pred = svr_model.predict(X_test)
    # Визуализация результатов
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test,   y_pred,   color='darkorange',
label='Predicted vs Actual')
    plt.plot([y_test.min(),              y_test.max()],
[y_test.min(), y_test.max()], 'k--', lw=2)
    plt.xlabel('Actual Prices')
    plt.ylabel('Predicted Prices')
    plt.title('SVR Predictions of Apartment Prices in
Moscow')
    plt.legend()
    plt.show()
```

На графике выше (рисунок 33) показано сравнение между фактическими ценами на квартиры и ценами, предсказанными моделью SVR для тестовой выборки. Точки, обозначенные оранжевым цветом, представляют собой предсказания модели по сравнению с реальными значениями цен. Чёрная пунктирная линия представляет идеальное соответствие, когда предсказанные значения равны фактическим.

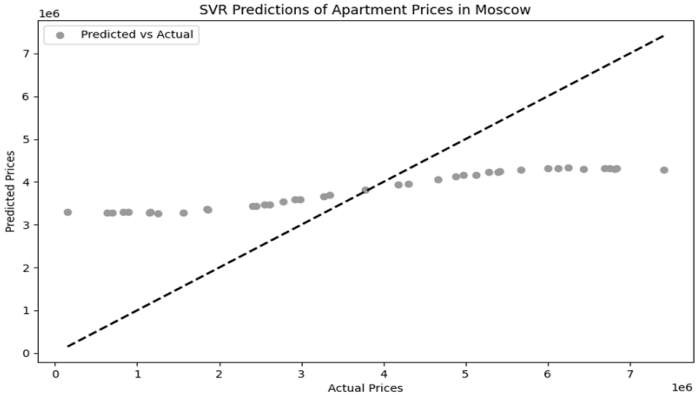

Рисунок 33. График SVR

График показывает, что модель SVR в целом смогла адекватно аппроксимировать цены, хотя и с некоторыми отклонениями. На основе этого графика можно сделать выводы о точности и эффективности использования SVR в задачах прогнозирования цен на недвижимость,

учитывая влияние различных факторов, таких как площадь квартиры, расстояние до центра и этаж.

3.6.3 Кластеризация

Кластеризация - это один из неконтролируемых методов в машинном обучении. Основная цель кластеризации распределить данные по группам (кластерам) таким образом, чтобы каждый экземпляр данных оказался в наиболее подходящем для него кластере. При этом нет необходимости заранее присваивать данным метки или оценки достоверности. Метод кластеризации имеет широкий спектр применения в реальной жизни, например, группировка клиентов и рынков для более эффективного взаимодействия с целевой аудиторией, сегментация изображений для упрощения их обработки и анализа, анализ социальных сетей для понимания интересов и поведения пользователей, группировка результатов поисковой системы для улучшения релевантности выдачи, обнаружение аномалий для выявления нетипичных ситуаций и предотвращения нежелательных событий.

Рассмотрим пример с набором данных, чтобы понять высокоуровневую концепцию, лежащую в основе кластеризации машинного обучения (рисунок 34).

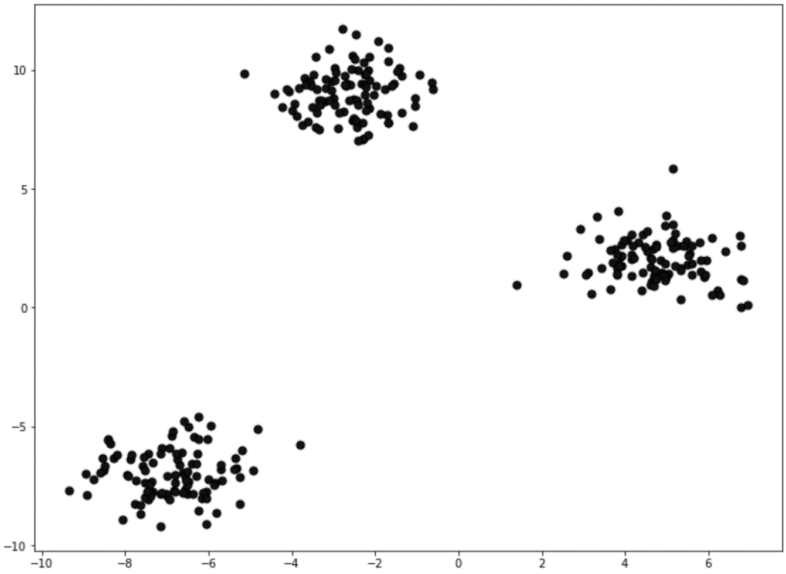

Рисунок 34. Набор данных

На первый взгляд представленные данные представляют собой набор точек без меток или информации о достоверности, каждая из которых имеет свою координату x и y. Задача нашего алгоритма машинного обучения - попытаться самостоятельно выявить структуру этих данных и распределить точки по группам. Результат кластеризации данных, которые представлены в виде разрозненных точек, может выглядеть следующим образом (рисунок 35).

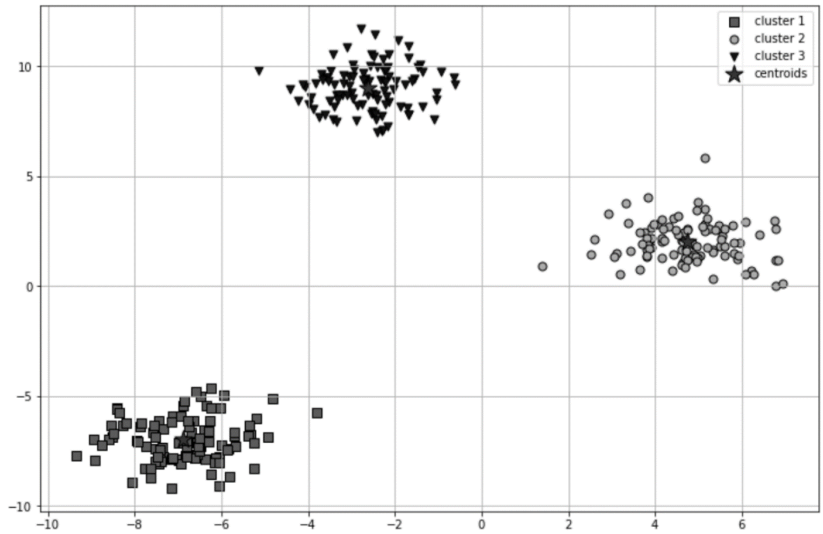

Рисунок 35. Результат кластеризации

На представленном рисунке визуализации видно, что точки данных, которые модель машинного обучения считает «похожими», сгруппированы в один кластер. Таким образом модель машинного обучения хорошо справилась с двумя задачами: определением необходимого количества кластеров и назначением каждой точки данных наиболее вероятному кластеру.

В машинном обучении существует множество алгоритмов кластеризации. Рассмотрим четыре наиболее распространённых из них:

1. **Кластеризация на основе центроида**. В машинном обучении существует метод кластеризации на основе центроидов. Суть этого метода заключается в том, что алгоритм случайным образом инициализирует определённое количество центроидов, которые должны быть заданы заранее. Далее происходит итерационный процесс, цель которого - найти

оптимальное положение для каждого центроида, чтобы минимизировать расстояние между точками данных и самим центроидом. Важно определиться с количеством центроидов до начала итерационного процесса.

2. **Кластеризация на основе распределения**. Это метод кластеризации, при котором точки данных группируются в кластеры на основании вероятности из определённого распределения, а не сходства или расстояния. То есть точки данных распределяются по кластерам в зависимости от вероятности принадлежности каждого элемента к определённому вероятностному распределению, обычно - гауссовому. Но также могут быть использованы и другие распределения вероятностей, например, биномиальное.

3. **Кластеризация на основе плотности**. Это алгоритм кластеризации, который группирует точки данных, находящиеся в одном регионе с высокой плотностью. Этот метод подходит для точек данных произвольной формы, расположенных близко друг к другу в плотной области.

4. **Иерархическая кластеризация**. Это алгоритм машинного обучения, который позволяет группировать точки данных в кластеры. Анализируется расстояние между каждой точкой данных и её окрестностями: точки данных, расстояние между которыми минимально, группируются в одну группу. Выделяют два подхода кластеризации на основе иерархии:

Агломеративный подход. Он начинается с того, что мы рассматриваем каждую точку данных как отдельный кластер. Затем происходит объединение кластеров таким образом, чтобы в конце всё рассматриваемое множество данных представляло собой один большой кластер.

Разделяющий подход. Предполагается, что все точки данных принадлежат одному большому кластеру. Цель этого подхода - разделить этот кластер на более мелкие на основе определённых пороговых значений расстояния или до тех пор, пока данные не будут разделены окончательно.

Рассмотрим различные типы алгоритмов кластеризации машинного обучения, которые основаны на вышеупомянутых подходах. Начнём с **алгоритма k-средних**.

Кластеризация k-средних - это один из алгоритмов, используемых в машинном обучении, который применяет подход кластеризации на основе центроида. Среди различных алгоритмов машинного обучения, обычно применяемых для кластеризации данных, кластеризация методом k-средних, вероятно, является наиболее популярным методом. Это объясняется тем, что он, как правило, показывает хорошие результаты, а также его концепция легко поддаётся объяснению.

Теперь реализуем кластеризацию k-средних на Python. Для этого воспользуемся библиотекой Scikit-learn и в частности экземпляром KMeans из модуля sklearn.cluster. Но сначала создадим набор данных и визуализируем его (рисунок 36).

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
# create dataset
X, y = make_blobs(
   n_samples=300, n_features=2,
   centers=3, cluster_std=1,
   shuffle=True, random_state=42
```

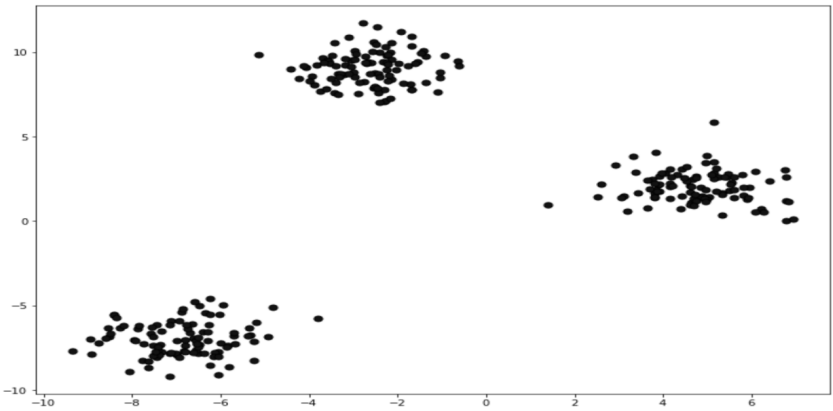

Рисунок 36. Набор данных

Как можно видеть из рисунка визуализации, наш набор данных включает в себя 300 точек данных. Исходя из расположения этих точек данных, мы предполагаем, что метод кластеризации по k-средним значениям создаст три кластера. Но проблема заключается в том, что необходимо заранее определить количество центроидов. В нашем случае мы знаем, что их должно быть 3, однако в реальных сценариях часто неизвестно, сколько центроидов надо установить. Чтобы разобраться с этой трудностью, мы используем метод Elbow.

```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
errors = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
```

```
    errors.append(kmeans.inertia_)
fig = plt.figure(figsize=(12, 9))
plt.plot(range(1, 10), errors)
plt.grid(True)
plt.xlabel('Error', fontsize=18)
plt.ylabel('No. of clusters', fontsize=16)
plt.title('Elbow curve',fontsize=18)
```

Метод elbow позволяет чётко определить, что оптимальное количество кластеров в данном случае – 3 (рисунок 37). Далее необходимо инициализировать и обучить новую модель кластеризации k-средних. После этого следует визуализировать результаты кластеризации (рисунок 38), а также обозначить местоположение каждого центроида.

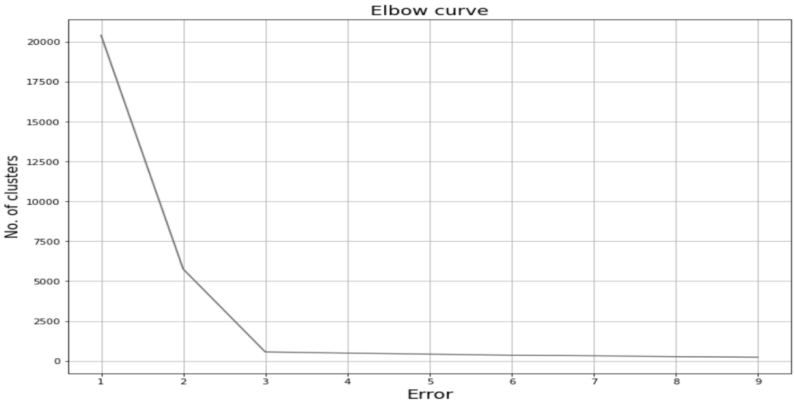

Рисунок 37. Применение метода elbow

```
# Initialize and train k-means model
km = KMeans(n_clusters=3)
y_kmeans = km.fit_predict(X)
# plot the clusters and centroids
plt.figure(figsize=(12,8))
plt.scatter(
    X[y_kmeans == 0, 0], X[y_kmeans == 0, 1],
    s=50, c='green',
    marker='s', edgecolor='black',
    label='cluster 1'
)
plt.scatter(
    X[y_kmeans == 1, 0], X[y_kmeans == 1, 1],
```

```
        s=50, c='orange',
        marker='o', edgecolor='black',
        label='cluster 2'
    )
    plt.scatter(
        X[y_kmeans == 2, 0], X[y_kmeans == 2, 1],
        s=50, c='blue',
        marker='v', edgecolor='black',
        label='cluster 3'
    )
    plt.scatter(
        km.cluster_centers_[:,                          0],
km.cluster_centers_[:, 1],
        s=250, marker='*',
        c='red', edgecolor='black',
        label='centroids'
    )
    plt.legend(scatterpoints=1)
    plt.grid()
    plt.show()
```

Из представленной визуализации видно, что алгоритм кластеризации k-средних успешно разделил наши данные на три группы.

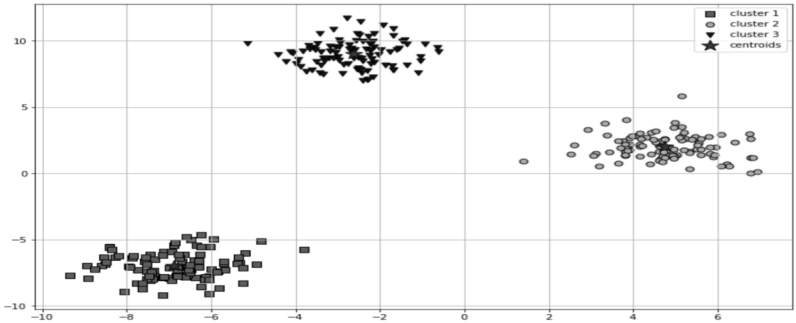

Рисунок 38. Результаты кластеризации с помощью алгоритма k-средних

Алгоритм агломеративной кластеризации в машинном обучении - это метод классификации данных, основанный на иерархии, который также может быть использован для группировки точек данных. Преимущество этого алгоритма заключается в том, что нет необходимости заранее определять количество кластеров, если это не требуется. Однако, хотя это и

не обязательно, качество кластеризации могло бы быть улучшено, если задать оптимальное количество кластеров на старте. Существует метод, позволяющий определить оптимальное количество кластеров при агломеративном методе, о котором мы расскажем подробнее далее.

Алгоритм работает следующим образом: каждой точке данных присваивается статус кластера. В результате первоначальных вычислений с данными, состоящими из четырёх точек, будет создано четыре кластера. Затем, после итеративного объединения точек, все данные будут включены в один большой кластер.

Чтобы лучше понять работу алгоритма, предположим, набор данных содержит четыре точки, как показано ниже (рисунок 39).

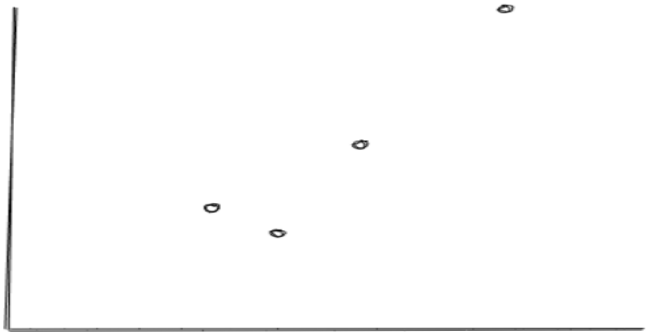

Рисунок 39. Набор данных

На первом этапе каждая точка данных будет помещена в собственный кластер, таким образом, если есть N точек данных, то изначально мы получим N кластеров (рисунок 40).

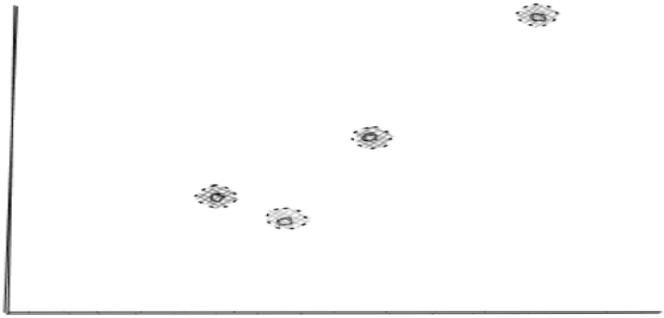

Рисунок 40. N кластеры

На следующем этапе две ближайшие точки данных будут объединены, таким образом после этой итерации будет на один кластер меньше - N-1 (рисунок 41).

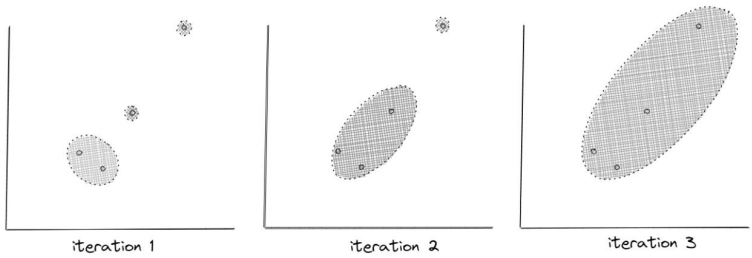

Рисунок 41. Применение итерации

Описанные шаги будут повторяться итеративно до тех пор, пока не сформируется единый кластер. После этого можно будет визуализировать историю слияний на каждой итерации с помощью дендрограммы. Алгоритм выбирает, какие точки данных будут объединены на каждой итерации. Обычно используется расстояние между двумя точками данных в разных кластерах. Для вычисления этого расстояния агломеративный подход использует две общие метрики:

- одинарная связь: расстояние вычисляется по ближайшим точкам данных между кластерами;
- полная увязка: расстояние вычисляется по самым удаленным точкам данных между кластерами.

Кроме того, применяются центроидная привязка (расстояние измеряется от центроида каждого кластера) и привязка к уорду (расстояние определяется так, чтобы разница между объединяемыми кластерами была минимизирована).

Выплним реализацию агломеративного алгоритма с помощью Python на том же наборе данных, который применялся в двух предыдущих алгоритмах кластеризации машинного обучения. Первым шагом визуализируем дендрограмму истории слияний для нашего случая, чтобы определить оптимальное число кластеров (рисунок 42).

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
```

```
import scipy.cluster.hierarchy as sch
plt.figure(figsize=(12, 8))
dendrogram = sch.dendrogram(sch.linkage(X,
method='ward'))
```

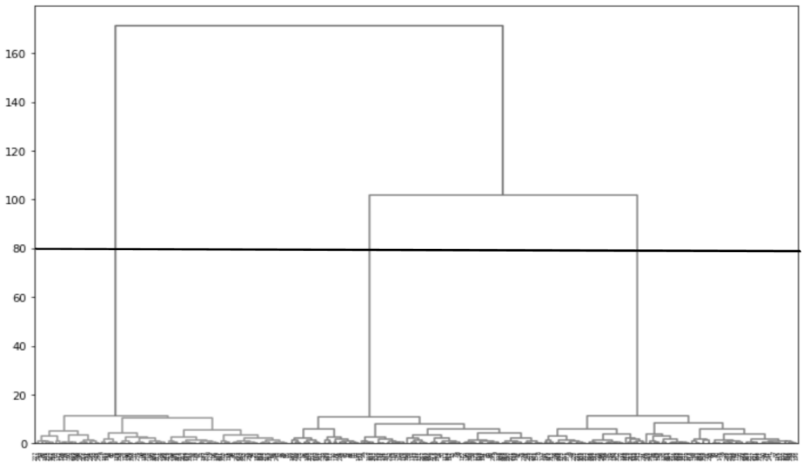

Рисунок 42. Дендограмма

Исходя из представленной визуализации, можно выбрать точку отсечения на уровне 80, после чего провести горизонтальную линию и определить, что оптимальное количество кластеров будет равно 3.

Теперь создадим экземпляр нашей агломеративной модели, которая будет состоять из трёх кластеров, и обучим её на нашем наборе данных, после этого мы сможем визуализировать результат кластеризации (рисунок 43).

```
def plot_scatter(X, labels, target, color,
marker_shape, legend):
    plt.scatter(
    X[labels == target, 0], X[labels == target, 1],
    s=50, c=color,
    marker=marker_shape, edgecolor='black',
    label=legend
    )
    def plot_agglo(agglo, X, label=True, ax=None):
    plt.figure(figsize=(12,8))
    ax = ax or plt.gca()
    labels = agglo.fit_predict(X)
```

```
        if label:
            plot_scatter(X, labels, 0, 'green', 's',
'cluster_1')
            plot_scatter(X, labels, 1, 'orange',
'o','cluster_2')
            plot_scatter(X, labels, 2, 'blue',
'v','cluster_3')
        ax.axis('equal')
        plt.grid(True)
    agglo = AgglomerativeClustering(n_clusters=3,
affinity='euclidean', linkage='ward')
    plot_agglo(agglo, X)
```

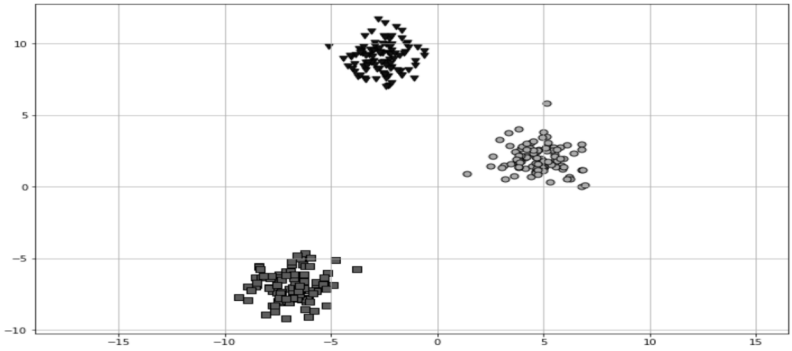

Рисунок 43. Результат кластеризации с помощью алгоритма агломеративной кластеризации

Одно из преимуществ использования агломеративного алгоритма кластеризации в машинном обучении заключается в том, что не нужно заранее задавать определённое количество кластеров. Вместо этого мы можем задать пороговое расстояние связи.

Если расстояние между двумя кластерами превышает пороговое, алгоритм не объединит эти кластеры. Чтобы наглядно продемонстрировать этот принцип, можно написать простой код на Python без необходимости задавать конкретное количество кластеров, а указать два разных значения порогового расстояния (например, 60 и 140), чтобы увидеть, как это влияет на результат кластеризации (рисунок 44).

```
    agglo_dist_60=AgglomerativeClustering(n_clusters=N
one, distance_threshold=60, affinity='euclidean',
linkage='ward')
```

```
   agglo_dist_140=AgglomerativeClustering(n_clusters=
None,  distance_threshold=140,  affinity='euclidean',
linkage='ward')
   plot_agglo(agglo_dist_60, X)
   plot_agglo(agglo_dist_140, X)
```

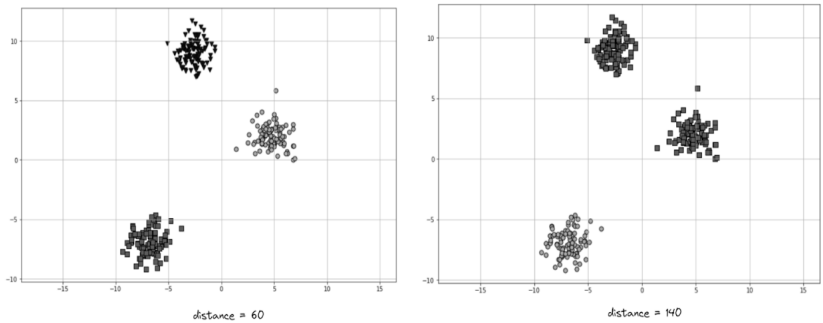

Рисунок 44. Варианты кластеризации с разными значениями порогового расстояния

Между двумя визуализациями есть одно очевидное различие: с левой стороны получилось 3 кластера, а с правой - только 2. Результат с 3 кластерами может быть лучше, но непонятно, почему возникло их разное количество, просто меняя пороговое значение расстояния.

Если посмотреть на дендрограмму выше, то станет понятно, что, если наша точка отсечения равна 60 и мы проводим горизонтальную линию поперёк графика, она пересекает 3 вертикальные линии, что соответствует результату слева. Между тем, если наша точка отсечения составляет 140, горизонтальная линия будет пересекать только 2 вертикальные. Это также соответствует результату нашей визуализации справа.

Итак, пороговое расстояние можно считать одним из гиперпараметров, которые нужно настроить, чтобы получить лучший результат кластеризации. Другими гиперпараметрами, которые стоит точно настроить, являются тип связи и показатели расстояния.

Пространственная кластеризация. Это метод, который позволяет группировать объекты на основе их плотности. Один из наиболее популярных алгоритмов пространственной кластеризации - **DBSCAN**.

Сначала алгоритм DBSCAN измеряет сходство между точками данных, после чего группирует наиболее похожие из них в один кластер. Особенность этого алгоритма в том, что он может находить кластеры

произвольной формы, а не только в виде гиперсфер, как многие другие алгоритмы, например, k-средних.

Для наглядности представим, что есть точки данных, которые по своей структуре похожи на наборы данных moon (рсиунок 45). Если мы попробуем применить к ним метод k-средних, результат будет похож на схему ниже.

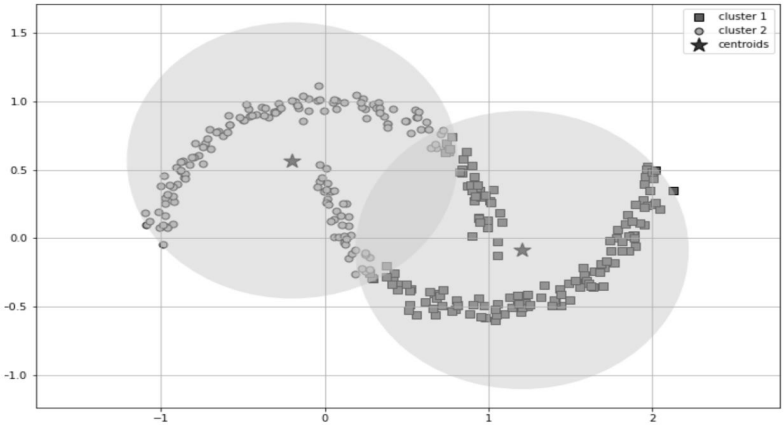

Рисунок 45. Набор данных

DBSCAN - это один из алгоритмов кластеризации, который может фиксировать кластеры произвольной формы.

Существуют два параметра, определяющие работу DBSCAN:

- eps - пороговое расстояние;
- min_samples - количество минимальных соседей в точке данных.

Для каждой точки данных алгоритм подсчитывает соседние точки данных в пределах эпсилона. Если количество соседей больше, чем min_samples, то точка данных - основная выборка, или выборка керна. Это означает, что она расположена в области с высокой плотностью данных. Все соседние точки выборки будут сгруппированы в один кластер.

Если какая-либо точка не является основной выборкой и не соседствует с основной выборкой, алгоритм кластеризации DBSCAN определит эту точку как выброс.

Реализуем алгоритм кластеризации машинного обучения DBSCAN на Python, используя библиотеку Scikit-learn. Для этого создадим набор данных (рисунок 46).

```
import numpy as np
from sklearn.cluster import DBSCAN
```

```
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler

X_moon, labels_true = make_moons(
    n_samples=300, noise = 0.05, random_state=0
)
```

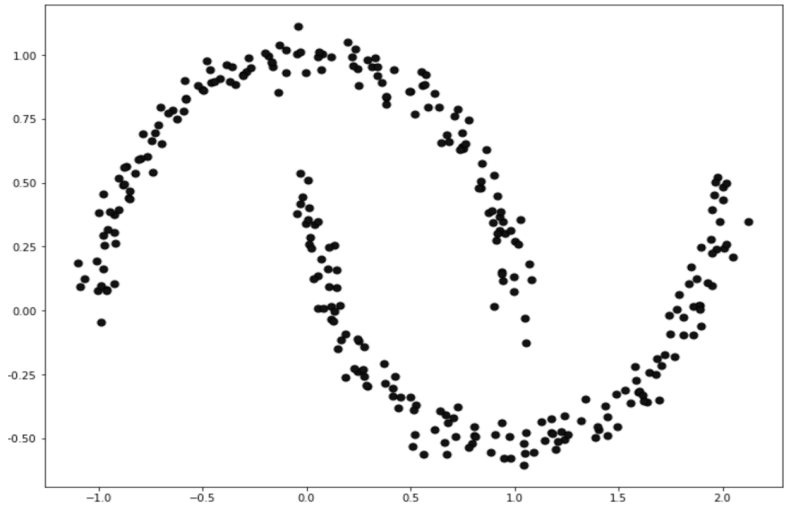

Рисунок 46. Набор данных

Для того чтобы реализовать алгоритм DBSCAN, сначала необходимо стандартизировать данные. Затем мы сможем вызвать класс DBSCAN из библиотеки Scikit-learn.

В нашем примере мы зададим пороговое расстояние eps, равное 0,3. Также установим количество минимальных соседей min_samples, равное 10.

```
X_moon = StandardScaler().fit_transform(X_moon)
db = DBSCAN(eps=0.3, min_samples=10).fit(X_moon)
```

После того как мы обучим модель DBSCAN на основе обучающих данных, мы сможем проверить, к какому кластеру принадлежит каждая из наших точек данных. Для этого существует метод labels_.

```
db.labels_
Output: array([0, 1, 1, ..., 0, 1, 0, 0, 0])
```

Результатом работы алгоритма DBSCAN будет массив, в котором каждый элемент соответствует одной из точек данных. Форма массива будет совпадать с количеством точек в выборке. Значение каждого элемента массива - это метка кластера, к которому принадлежит

соответствующая точка данных. В описанном примере использования DBSCAN алгоритм определил, что оптимальное количество кластеров равно двум, поэтому в массиве будет только два значения: 0 и 1.

Если в наборе данных есть выбросы, то DBSCAN определяет их как шумовые и присваивает им метку -1.

Визуализацию работы алгоритма можно увидеть ниже (рисунок 47).

Существует ещё один алгоритм кластеризации машинного обучения, который является расширением алгоритма DBSCAN - **HDBSCAN или иерархический DBSCAN**.

Алгоритм работает аналогично DBSCAN, но имеет одно отличие: в HDBSCAN не нужно заранее указывать значение параметра epsilon. Это становится возможным благодаря тому, что HDBSCAN пытается найти оптимальные кластеры на основе различных расстояний.

Таким образом, сохраняются все положительные качества DBSCAN. Кроме этого, HDBSCAN может идентифицировать кластеры с различной плотностью. Из характеристик HDBSCAN следует, что нужно определить только один параметр - это минимальный размер кластера.

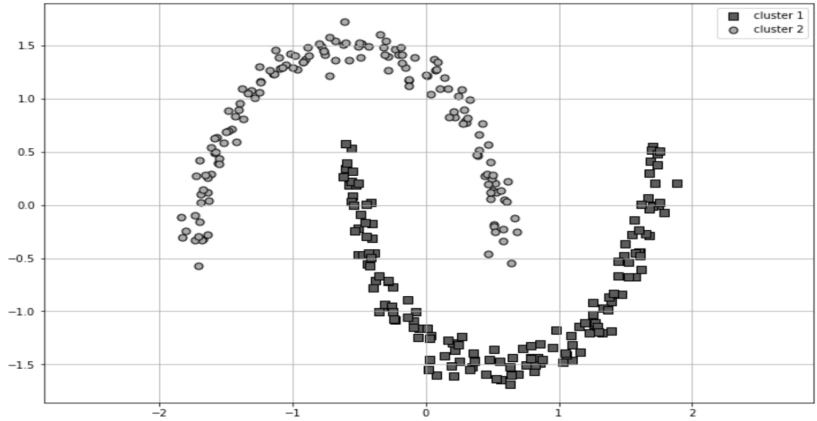

Рисунок 47. Результат кластеризации с помощью алгоритма DBSCAN

Для работы с алгоритмом существует библиотека на Python, которую можно установить через pip. API этой библиотеки похож на тот, что мы реализовали выше с помощью Scikit-learn. Ниже приведён пример того, как мы можем реализовать HDBSCAN.

```
import hdbscan
import numpy as np
from sklearn.preprocessing import StandardScaler
```

```
hdb = hdbscan.HDBSCAN(min_cluster_size=5,
gen_min_span_tree=True).fit(X_moon)
```

Хотя для корректной работы алгоритма иерархической кластеризации с древовидной связью HDBSCAN (HDBSCAN) и не требуется указывать фиксированное пороговое значение расстояния, также известное как «эпсилон» (epsilon), необходимо определить минимальный размер кластера. Если в процессе разделения получается кластер менее указанного размера, вместо создания нового кластера такие данные будут рассматриваться как выпадающие.

Определить принадлежность каждой точки данных к определённому кластеру можно с помощью метода label (рисунок 48).

```
hdb.labels_
Output: array([1, 0, 0, ..., 0, 1, 1, 1]
```

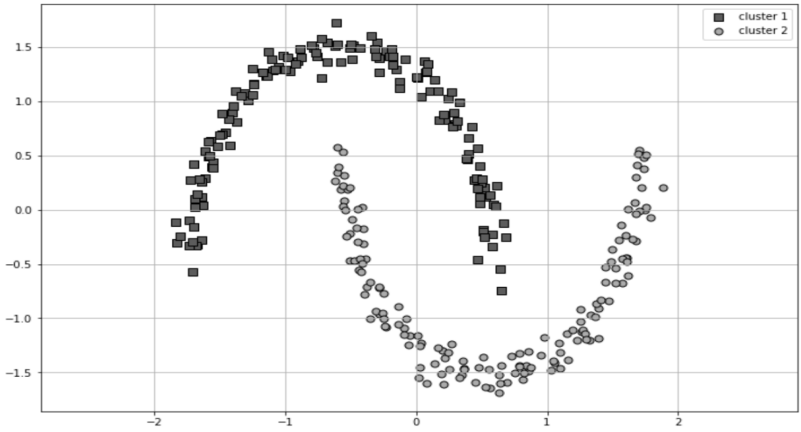

Рисунок 48. Результат кластеризации с помощью алгоритма HDBSCAN

Таким образом методы кластеризации машинного обучения представляют собой неконтролируемый метод обучения. К различным алгоритмам кластеризации относятся: кластеризация по k-средним, модель гауссовой смеси, агломеративная кластеризация и DBSCAN.

Метод кластеризации по k-средним хорошо работает, когда точки данных имеют форму круглых кластеров, и при этом его рабочий процесс относительно прост для понимания. В свою очередь, DBSCAN и модель гауссовой смеси более гибкие с точки зрения формы кластера.

Алгоритмы агломеративной кластеризации и DBSCAN подходят для случаев, когда не требуется заниматься поиском оптимального кластера для поставленной задачи.

Контрольные вопросы по главе 3

1. Охарактеризуйте библиотеку Pandas. Какими возможностями она обладает?
2. Охарактеризуйте библиотеку NumPy. Какими возможностями она обладает?
3. Охарактеризуйте библиотеку SymPy. Какими возможностями она обладает?
4. Охарактеризуйте библиотеку Matplotlib. Какими возможностями она обладает?
5. Охарактеризуйте библиотеку TensorFlow. Какими возможностями она обладает?
6. Что такое машинное обучение?
7. Дайте определение «классификация».
8. Какие алгоритмы для задач классификации вы знаете?
9. Дайте определение «регрессия».
10. Какие виды регрессии вы знаете? Поясните каждый вид.
11. Дайте определение «кластеризация».
12. Какие вы знаете алгоритмы кластеризации?

Глава 4. ТЕХНОЛОГИИ РАБОТЫ СО СТРУКТУРИРОВАННЫМИ ДАННЫМИ

Структурированные данные - это данные, которые сформированы и отформатированы определённым образом. Для этого используют определённую схему или модель данных, которая обеспечивает их структуру. За счет своей структуры структурированные данные могут быть легко читаемыми и понятными как людям, так и машинам.

К структурированным данными обычно относятся базы данных и электронные таблицы. Каждому элементу данных в схеме присваивается определённое поле или столбец, а каждая запись или строка представляет конкретный экземпляр этих данных. В базе данных клиентов каждая запись может содержать поля для имени клиента, адреса, номера телефона и адреса электронной почты.

Структурированность означает соответствие модели данных, чёткую структуру, последовательный порядок, доступность для человека или компьютерной программы. Хранение структурированных данных осуществляется в базах данных с помощью схем. Они зачастую имеют табличную форму со строками и столбцами, которые чётко определяют атрибуты данных. Популярным инструментом для управления структурированными данными в базах является язык структурированных запросов SQL.

Благодаря своей организованной природе структурированные данные легко искать, запрашивать и анализировать с помощью различных инструментов. Это делает их идеальным форматом для приложений, основанных на данных, таких как бизнес-аналитика, машинное обучение и искусственный интеллект.

Примеры форматов, структурированных данных - реляционные базы данных, XML, CSV, HTML и JSON. Напротив, неструктурированные данные, такие как текстовые документы или изображения, не имеют предопределённой схемы или структуры, поэтому их сложно анализировать и интерпретировать.

4.1. Файлы и файловые системы

4.1.1 Файлы

Файл — это часть внешней памяти компьютера, которая имеет свой уникальный идентификатор и содержит определённые данные. Существуют программные (исполняемые) файлы, которые управляют работой

компьютера и файлы, содержащие различные данные, например, тексты, изображения.

Идентификатор файла состоит из его имени и расширения, при этом между именем файла и его расширением обычно ставится разделитель в виде точки. Расширение представляет из себя набор символов, которые указывают операционной системе, какую программу следует использовать при работе с этим файлом. На магнитных дисках файл представлен в виде набора кластеров. Организацию и управление файлами осуществляет файловая система.

В файлах хранится большое количество неструктурированных данных. Благодаря формату, эти данные упорядочены, их можно легко хранить, обрабатывать и использовать.

Формат файла представляет собой установленный способ организации данных в компьютерном файле. Существует множество форматов файлов, которые используются для эффективного хранения и извлечения информации. Рассмотрим особенности каждого из них, а также их назначение и сравним их.

Текстовый файл - это файл, который содержит упорядоченный набор буквенно-цифровых символов и служит контейнером для текстовых данных. Подобные файлы являются самыми простыми и важными. Они используются для хранения текстовых материалов и обмена ими. Текстовые файлы находят применение как в повседневной жизни обычных пользователей, так и при разработке программного обеспечения.

Различные типы текстовых форматов (.doc, .docx, .rtf, .pdf, .wpd и т.д.) служат для создания и редактирования текстовых документов (таблица 8). Текстовые файлы не содержат форматирования - оно применяется, когда открывается документ в программе.

Таблица 8 - Форматы текстовых файлов

Расширение	**Полная форма**	**Описание**
.txt	Обычный текст	Самый простой формат текстового файла, содержащий только символы ASCII и возврат каретки в отдельные строки.
.rtf	Формат расширенного текста	Более продвинутый формат текстового файла, который допускает базовое форматирование, такое как жирный шрифт, курсив и стили шрифтов.
.docx	Открытый XML-документ Word	Обычно используется Microsoft Word для хранения документов
.csv	Значения, разделенные запятыми	Простой формат для хранения табличных данных, где каждая строка представляет

		собой запись данных, а поля разделяются запятыми.
.doc	Документ Word	Используется для текстовой обработки документов, хранящихся в двоичном формате файла Microsoft Word
.wps	Документ WPS Office Word	Проприетарный формат файла документа, разработанный Kingsoft Office.
.wpd	Документ WordPerfect	Формат файла документа, связанный с WordPerfect, программой для обработки текстов.
.msg	Сообщение	Формат сообщений Microsoft Outlook; содержит сообщения электронной почты с форматированием, вложениями и другой информацией.

Для кодирования текстовых файлов в основном используются такие простые кодировки символов, как *UTF-8* и *ASCII*. Текстовые данные могут быть сохранены как в обычном, так и расширенном текстовом формате. Текстовые файлы могут применяться для хранения программных данных.

Изображения сохраняются в различных форматах (.jpeg, .png, .gif, .heif и других) (таблица 9). Эти файлы хранят данные о графических объектах в двоичном виде, причём особенности каждого формата определяют, как изображение будет сжиматься и храниться.

Таблица 9 - Форматы графических файлов

Расширение	Полная форма	Описание
.jpg	Объединенная группа экспертов по фотографии	Формат сжатия с потерями, который обычно используется для фотографий и других изображений с большим количеством деталей.
.png	Портативная сетевая графика	Формат сжатия без потерь, который обычно используется для изображений с четкими краями или текста.
.webp	Формат веб-изображений	Поддерживает сжатие изображений как с потерями, так и без них с поддержкой 24-битного цвета RGB.
.gif	Формат обмена графикой	Формат с ограниченным количеством цветов обычно используется для анимации и небольших изображений.
.tif	Формат файла изображения с тегами	Высококачественный формат, который обычно используется для профессиональной фотографии и печати.

.bmp	Растровое изображение	Несжатый формат, который обычно используется Microsoft Windows.
.eps	Инкапсулированный файл PostScript	Векторный формат, который обычно используется для печати графики

Аудиофайлы - это файлы, которые хранят цифровые аудиоданные в компьютерных системах. Формат аудиофайлов определяет структуру этих данных. Часть этой структуры, которая описывает сами аудиоданные (исключая метаданные), называется форматом кодирования звука. Часто аудиоданные сжимают, чтобы уменьшить размер файла. Этот процесс сжатия называется сжатием с потерями. Он позволяет уменьшить размер аудиофайла, но может привести к некоторой потере качества звука. Аудиоданные хранятся в аудиоформатах (.mp3, .aac, .wav и прочих), где необработанные данные закодированы, а для сжатия и распаковки используют кодек (таблица 10).

Таблица 10 - Форматы аудиофайлов

Расширение	**Полная форма**	**Описание**
.mp3	Аудиофайл MP3	Обычно используется для хранения и распространения музыки.
.snd	Звук	Общее расширение файла для звуковых файлов, часто ассоциируемое со звуковыми данными.
.wav	Аудиофайл WAVE	Обычно используется для хранения и записи аудио.
.ra	RealAudio	Это формат файла списка воспроизведения, который обычно используется для хранения и распространения списков воспроизведения.
.au	Аудио	Аудио
.aac	Расширенное кодирование звука	Используется в качестве модного дизайна звукового поля для упаковки виртуального звука и данных настройки.

Видеофайл представляет собой особым образом, структурированный набор данных, в котором информация закодирована и сжата с использованием специальных алгоритмов. Область разработки форматов хранения видео постоянно развивается, улучшая качество восприятия видеоконтента пользователями и предоставляя возможности для развития цифровых сервисов.

В настоящее время существует большое количество различных форматов видеофайлов. В основном, выделяют 5 форматов, которые имеют широкое распространение и заслуживают отдельного внимания (таблица 11).

Таблица 11 - Форматы аудиофайлов

Расширение	Полная форма	Описание
.mp4	Видеофайл MPEG-4	Формат мультимедийного контейнера, в котором обычно хранятся видео- и аудиоданные.
.3gp	Мультимедийный файл 3GPP	Формат мультимедийного контейнера, который обычно используется для мобильных телефонов.
.avi	Файл с чередованием аудио и видео	Старый формат мультимедийного контейнера, который по-прежнему поддерживается многими устройствами.
.mpg	Видеофайл MPEG	Старый формат сжатия видео, который все еще поддерживается некоторыми устройствами.
.mov	Apple QuickTime Movie	Формат, который обычно используется на устройствах Apple.
.wmv	Видеофайл Windows Media	Формат, который обычно используется устройствами Microsoft.

Форматы программных файлов играют решающую роль в запуске компьютерных приложений. Формат этих файлов определяет, как в них представлена информация, позволяя компьютерам понимать, что означают коды. К основным программным файлам относятся: .bat, .com и .exe. Файлы, используемые в программировании (.py, .c, .java, .js и прочие), хранят код, который компьютер может запустить путем компиляции или интерпретации (таблица 12).

Таблица 12 - Форматы аудиофайлов

Расширение	Полная форма	Описание
.c	Файл исходного кода C /C ++	Язык программирования общего назначения, разработанный Деннисом Ричи в Bell Labs в период с 1969 по 1972 год.
.cpp	Файл исходного кода C ++	Язык программирования общего назначения, разработанный Бьярне Страуструпом как расширение языка программирования C.
.java	Файл исходного кода Java	Язык программирования, созданный Sun Microsystems, который в настоящее время принадлежит Oracle Corporation.
.py	Скрипт на Python	Язык программирования был разработан Гвидо ван Россумом и впервые выпущен в 1991 году.

.js	Javascript	Язык сценариев, который в основном используется для придания интерактивности веб-страницам.
.ts	TypeScript	Надмножество JavaScript, добавляющее необязательную статическую типизацию.
.cs	Файл исходного кода C #	Язык программирования, разработанный Microsoft как часть .NET Framework.
.swift	Файл с исходным кодом Swift	Язык программирования, разработанный Apple для разработки приложений iOS, macOS, watchOS, tvOS и Linux
.dta	Файл определения типа документа	Формат хранения данных, обычно используемый статистической программой Stata.
.pl	Perl Скрипт	Язык программирования, разработанный Ларри Уоллом в Калифорнийском университете в Санта-Крузе в начале 1980-х годов.
.sh	Скрипт оболочки Bash	Язык сценариев оболочки, обычно используемый для автоматизации задач в Unix-подобных операционных системах
.bat	Пакетный файл	Формат пакетного файла, используемый для автоматизации задач в системах Windows; содержит серию команд, выполняемых интерпретатором команд.
.com	Командный файл	COM-файл - это формат исполняемого файла, используемый для программ в старых системах Windows. COM-файлы имеют ограниченную функциональность по сравнению с современными форматами.
.exe	Исполняемый файл	Исполняемый файл - это тип компьютерного файла, который содержит скомпилированный код, который может быть запущен непосредственно операционной системой. Исполняемые файлы обычно используются для запуска программ.

Сжатые файлы - важная часть цифровой среды, позволяющая сократить объёмы данных и сэкономить место в хранилище. Для передачи данных между компьютерами их сжимают: они хранятся в сжатом формате (.rar, .tar, .iso, .7z и других), но перед использованием файл надо распаковать. Такие файлы называют архивными (таблица 13).

Таблица 13 - Форматы аудиофайлов

Расширение	Полная форма	Описание
.rar	Сжатый архив WinRAR	Фирменный файловый архиватор, разработанный Евгением Рошалем.
.zip	Архивированный файл	Формат сжатия данных без потерь, который упаковывает несколько файлов в один архивный файл.
.hqx	BinHex	Формат преобразования двоичного файла в текст для Macintosh, часто используемый для передачи двоичных файлов по электронной почте.
.arj	Архивировано Робертом Юнгом	Формат сжатия файлов, аналогичный ZIP и RAR, используемый для сжатия и архивирования файлов.
.tar	Сжатый архивный файл	Этот формат архивирования файлов объединяет несколько файлов в один архивный файл.
.arc	Архивный файл ARC	Файл ARC - это формат архивного файла, используемый для сжатия и хранения файлов. ARC - устаревший формат, который был заменен ZIP и другими более новыми вариантами.
.sit	архивный файл StuffIt	Файл SIT - это формат архивного файла, используемый в системах Macintosh. SIT похож на ARC, но специфичен для компьютеров Mac.
.gz	Сжатый файл GZIP	Файл GZ - это формат файла, созданный с помощью сжатия gzip. Gzip уменьшает размер файлов для хранения и передачи.
.z	Сжатый файл	Файл Z - это сжатый формат файла, связанный с программой сжатия “compress” в системах Unix.

Файлы веб-страниц - особый формат, который используется исключительно для хранения содержимого и представления его в интернете. Информация собирается в этих файлах воедино, в них же она

размещается, и благодаря им становится возможным перемещаться по веб-сайтам. Веб-страницы (.xps, .asp, .html, .css и так далее) обычно содержат программные скрипты для статичных или динамичных веб-страниц, информацию о веб-сайте в целом и отдельных веб-страницах (таблица 14). Эти файлы могут считываться и отображаться браузерами.

Таблица 14 - Форматы аудиофайлов

Расширение	Полная форма	Описание
.html	Файл языка гипертекстовой разметки	HTML - это стандартный язык разметки для создания веб-страниц.
.htm	Файл языка гипертекстовой разметки	Формат документа на языке гипертекстовой разметки (HTML) с менее распространенным расширением файла; идентичен файлам .html.
.xhtml	Расширяемый файл языка гипертекстовой разметки	Это язык разметки, сочетающий HTML с XML.
.asp	Страница активного сервера	Технология веб-разработки, которая позволяет создавать динамические веб-страницы с использованием серверных сценариев.
.css	Каскадная таблица стилей	Это язык таблицы стилей, используемый для описания представления веб-страницы.
.aspx	Расширенный файл страницы активного сервера	Это позволяет создавать динамические веб-страницы, используя серверные скрипты в ASP.NET.
.rss	Подробное описание сайта	Это формат веб-ленты, который позволяет пользователям подписываться на обновления с веб-сайтов.

4.1.2 Файловые системы

Вся информация обычно записывается, хранится и обрабатывается на различных цифровых носителях в виде файлов. В зависимости от типа файла, он кодируется в виде расширений - например, exe, doc или pdf. После этого происходит их открытие и обработка в соответствующем программном обеспечении. Как правило, на этом этапе мало кто задумывается, как на самом деле происходит хранение и обработка цифрового массива на носителе.

Операционная система воспринимает физический диск, где хранится информация, не как единое целое, а как набор кластеров размером от 512 байт и более. Драйверы файловой системы, в свою очередь, организуют эти кластеры в файлы и каталоги, которые тоже представляют собой файлы, но содержат список других файлов в этом каталоге. Они же отслеживают, какие кластеры используются, какие свободны, а какие помечены как неисправные.

Запись файлов большого объёма приводит к необходимости фрагментации - когда файлы не сохраняются как целые единицы, а делятся на фрагменты, которые записываются в отдельные кластеры. Ячейка кластера состоит из одного байта, и информация о всех фрагментах, как составляющей одного файла, хранится в файловой системе.

Файловая система обеспечивает связь между носителем информации (хранилищем) и прикладным программным обеспечением и предоставляет программе функционал взаимодействия программ API, чтобы организовать доступ к конкретным файлам. Когда программа обращается к файлу, она располагает данными о его имени, размере и атрибутах. Остальную информацию, относящуюся к типу носителя и структуре хранения данных, она получает от драйвера файловой системы.

Функция драйверов ФС на физическом уровне заключается в оптимизации записи и считывания отдельных частей файлов для ускоренной обработки запросов, фрагментации и «склеивания» информации, хранящейся в ячейках. На концептуальном уровне этот алгоритм был реализован в большинстве популярных файловых систем как иерархическая структура представления метаданных (B-trees). Использование этой технологии приводит к снижению времени выполнения самых длительных дисковых операций -позиционирования головок при чтении произвольных блоков. Таким образом, удаётся не только ускорить обработку запросов, но и продлить срок службы HDD. Однако при переходе к SSD, где принцип записи, хранения и считывания информации отличается от применяемого в жёстких дисках, существует ряд нюансов, связанных с выбором оптимальной файловой системы для конкретных задач.

Файловая система -это совокупность условий и правил, определяющих способ организации файлов на носителях информации. Драйвер ФС организует взаимодействие между хранилищем, операционной системой и прикладным программным обеспечением. Правильный выбор файловой системы для конкретных пользовательских задач влияет на скорость обработки данных, принципы распределения и другие функциональные возможности, необходимые для стабильной работы компьютерных систем.

Основные функции файловой системы:

1. Размещение и упорядочивание на носителе данных в виде файлов.

2. Определение максимально поддерживаемого объёма данных на носителе информации.

3. Создание, чтение и удаление файлов.

4. Назначение и изменение атрибутов файлов, таких как размер, время создания и изменения, владелец и создатель файла, доступ только для чтения, скрытый файл, временный файл, архивный, исполняемый, максимальная длина имени файла и т.п.

5. Определение структуры файла.

6. Поиск файлов.

7. Организация каталогов для логической организации файлов.

8. Защита файлов при системном сбое.

9. Защита файлов от несанкционированного доступа и изменения их содержимого.

Функционал файловой системы направлен на выполнение ряда задач, таких как присвоение имён файлам, предоставление программного интерфейса для работы с файлами различным приложениям, отображение логической модели файловой системы на физическую организацию хранилища данных, обеспечение устойчивости файловой системы к сбоям питания, ошибкам аппаратных и программных средств, а также хранение параметров файла, которые необходимы для корректного взаимодействия с другими объектами системы -ядром, приложениями и прочими.

В многопользовательских системах реализуется задача защиты файлов от несанкционированного доступа и обеспечения совместной работы. Когда один из пользователей открывает файл, для других пользователей этот же файл становится временно доступным только для чтения.

Информация о файлах хранится в специальных областях раздела (томах). Структура справочников зависит от типа файловой системы. Справочник файлов связывает числовые идентификаторы уникальных файлов с дополнительной информацией о них и непосредственным содержимым файла, которое хранится в другой области раздела.

Существует три основных вида операционных систем, используемых для управления любыми информационными устройствами: Windows компании Microsoft, macOS разработки Apple и операционные системы с открытым исходным кодом на базе Linux. Все они, для взаимодействия с физическими носителями, используют различные типы файловых систем. В случае Winodws всё выглядит достаточно просто - NTFS на всех дисковых разделах и FAT32 (или NTFS) на флеш-накопителях. Если установлен NAS

(сервер для хранения данных на файловом уровне) и в нем используется какая-то другая файловая система, на это практически не обращают внимания. Достаточно просто подключить устройство к серверу по сети и загрузить файлы.

На мобильных гаджетах с ОС Android часто установлена файловая система версии ext4 во внутренней памяти и FAT32 - на картах microSD. Определить, какая именно используется файловая система на устройствах пользователей продукции Apple, зачастую невозможно. Пользователи просто видят красивые значки папок и файлов в графическом интерфейсе, не задумываясь о технических характеристиках системы. Кроме того, на мобильных устройствах Apple могут использоваться файловые системы таких типов как HFS+, HFSX, APFS, WTFS или любая другая. Для пользователя существуют лишь красивые значки папок, поэтому информацию о специфике файловой системы получить сложно.

Более богатый выбор у ОС Linux. Однако настройка и использование определенного типа файловой системы требует хотя бы минимальных навыков программирования.

Файловые системы Windows. Исходный код файловой системы FAT (*File Allocation Table*) был разработан по личной договорённости владельца Microsoft Билла Гейтса с первым наёмным сотрудником компании Марком Макдональдом в 1977 году. Главной задачей FAT являлась работа с данными в операционной системе Microsoft 8080/Z80 на базе платформы MDOS/MIDAS.

Эта файловая система претерпела несколько модификаций: FAT12, FAT16 и, наконец, FAT32, которая на сегодняшний момент используется в большинстве внешних накопителей. Ключевым отличием каждой версии было преодоление ограниченного объёма доступной для хранения информации.

Впоследствии были созданы ещё две более продвинутые системы обработки и хранения данных -NTFS (*New Technology File System*) и ReFS (*Resilient File System*).

Файловая система FAT. FAT12, FAT16 и FAT32 - это версии файловой системы, в которых числа обозначают количество бит, используемых для перечисления блока файловой системы. FAT32 является фактическим стандартом и устанавливается на большинстве видов сменных носителей по умолчанию. Эта версия файловой системы (ФС) применяется не только на современных моделях компьютеров, но и в устаревших устройствах и консолях, снабжённых разъёмом USB.

Пространство FAT32 логически разделено на три сопредельные области:

- зарезервированный сектор для служебных структур,
- табличная форма указателей,
- непосредственная зона записи содержимого файлов.

Недостатком стандарта FAT32 можно считать ограничение размера файлов на диске до 4 ГБ и всего раздела в пределах 8 ТБ. По этой причине данная ФС чаще всего используется в USB-накопителях и других внешних носителях информации. Для установки последней версии ОС Microsoft Windows 10 на внутренний носитель потребуется более продвинутая файловая система.

Чтобы устранить ограничения, присущие FAT32, корпорация Microsoft разработала exFAT (расширенная таблица размещения файлов). Эта новая ФС очень похожа на своего предшественника, но позволяет пользователям хранить файлы намного большего размера, чем 4 гигабайта. Также в ней значительно снижено число перезаписей секторов, отвечающих за непосредственное хранение информации. Это важная функция для твердотельных накопителей в связи с необратимым изнашиванием ячеек после определённого количества операций записи. ExFAT совместим с операционными системами Mac, Android и Windows (для Linux понадобится вспомогательное ПО).

Файловая система NTFS. Стандарт файловой системы NTFS был разработан для того, чтобы исправить недостатки предыдущих файловых систем. Впервые он был применён в операционной системе Windows NT в 1995 году и в настоящее время является основной файловой системой для Windows. NTFS увеличила допустимый предел размера файлов до 16 гигабайт и поддерживает разделы диска объёмом до 16 эксабайт (10^{18} байт).

Данная система файлового хранения данных использует метод шифрования Encryption File System. Метод «прозрачного шифрования» разграничивает доступ к данным для разных пользователей и защищает содержание файлов от несанкционированного доступа. Также NTFS позволяет использовать расширенные имена файлов и поддерживает многоязычность в стандарте юникода UTF, включая файлы в формате кириллицы.

Для повышения надёжности работы жёсткого диска или внешнего накопителя в NTFS включено приложение проверки на ошибки файловой системы chkdsk. Однако использование данной функции может негативно повлиять на производительность устройства.

Файловая система ReFS. ReFS -это файловая система, разработанная компанией Microsoft, которая отличается высокой отказоустойчивостью и доступна для серверов Windows 8 и 10.

Её архитектура организована в виде B+ -tree. Файловая система обладает рядом особенностей для обеспечения безопасности и надёжности данных:

- технология Copy-on-Write (CoW) не позволяет изменять метаданные без предварительного копирования;
- новая версия данных записывается на свободное дисковое пространство, а не поверх существующих файлов;
- при модификации метаданных система создаёт новую копию, сохраняя её в свободном пространстве диска, и затем связывает старую версию метаданных с новой.

Такой подход повышает надёжность хранения данных и позволяет осуществлять их быстрое и лёгкое восстановление.

Файловые системы macOS. Для операционной системы macOS компания Apple использует собственные разработки файловых систем: HFS+, которая представляет собой усовершенствованную версию HFS, ранее используемой на компьютерах Macintosh, и её более современную версию APFS.

Стандарт HFS+ применяется на всех устройствах под управлением продукции Apple, включая компьютеры Mac, iPod и Apple X Server.

Кроме того, в расширенных серверных продуктах компании используется кластерная файловая система Apple Xsan, созданная на основе файловых систем StorNext и CentraVision. Эта файловая система отвечает за хранение файлов и папок, а также информации Finder о просмотре каталогов, положении окна и прочем.

Файловые системы Linux. В отличие от ОС Windows и macOS, которые ограничивают выбор файловой системы предустановленными вариантами, в Linux есть возможность использовать несколько файловых систем (ФС), каждая из которых оптимизирована для определённых задач. В Linux файловые системы используются не только для работы с файлами на диске, но и для других целей: они могут хранить данные в оперативной памяти или позволять доступ к конфигурации ядра во время работы системы. Все эти файловые системы включены в ядро Linux и могут быть использованы в качестве корневой.

Основные файловые системы, используемые в дистрибутивах Linux: Ext2, Ext3, Ext4, JFS, ReiserFS, XFS, Btrfs и ZFS.

Файловая система Ext. Одной из наиболее популярных файловых систем является Ext, которая первоначально была разработана для операционной системы Minix. Она содержит максимальное количество функций и является наиболее стабильной, так как её кодовая база редко меняется. Начиная с версии Ext3 в ней используется функция журналирования. В настоящее время эта файловая система установлена во всех дистрибутивах Linux.

Файловая система JFS. JFS или Journaled File System, также известная как Journaled filesystem, была разработана IBM как альтернатива файловым системам Ext. Сейчас она применяется там, где необходима высокая стабильность и минимальное потребление ресурсов, особенно в многопроцессорных компьютерах. JFS хранит только метаданные в журнале, позволяя восстанавливать старые версии файлов после сбоев.

Файловая система ReiserFS. ReiserFS -ещё одна альтернатива Ext3, разработанная специально для Linux. Эта файловая система обладает рядом преимуществ. Например, динамический размер блока позволяет упаковывать несколько небольших файлов в один блок, что предотвращает фрагментацию и улучшает работу с небольшими файлами. Однако при отключении энергии существует риск потери данных.

Файловая система XFS. Для работы с файлами большого размера подходит файловая система XFS. Она поддерживает диски до двух терабайт и работает быстрее с большими файлами. XFS обладает отложенным выделением места, увеличением разделов на лету и незначительным размером служебной информации. Однако она не позволяет уменьшить размер диска, сложно восстанавливает данные и может привести к потере файлов при аварийном отключении питания.

Файловая система Btrfs. Btrfs (или B-Tree File System) является легко администрируемой файловой системой, обладающей высокой отказоустойчивостью и производительностью. Она часто используется в качестве файловой системы по умолчанию в OpenSUSE и SUSE Linux. Другие файловые системы можно также использовать в Linux, но обычно их не применяют в качестве корневой файловой системы, так как они не предназначены для этого.

4.1.3 Работа с файлами

Файлы и папки в операционной системе Windows — это специальные объекты с набором определённых свойств, которые используются для

удобного упорядочивания информации в файловой системе компьютера с целью облегчения быстрого доступа к необходимым данным.

Операция создания папок и файлов. Чтобы создать папку, нужно открыть Проводник на нужном месте жёсткого диска, щёлкнуть правой кнопкой мыши (ПКМ) по свободному месту и выбрать в контекстном меню «*Создать*», а затем «*Папку*». После этого нужно ввести имя папки и нажать Enter или щёлкнуть по свободному месту.

С файлами ситуация немного другая. Обычно они создаются непосредственно в программе, которая с ними работает. Например, текстовый файл в программе «Блокнот», а рисунки - в графическом редакторе Paint, установленном в системе по умолчанию. Однако можно пойти другим путём и создать пустой файл некоторых типов, не запуская соответствующую программу. Для этого нужно щёлкнуть ПКМ по свободному пространству в Проводнике и выбрать «*Создать*» и нужный тип файла, например, «*Текстовый документ*». Затем требуется ввести на клавиатуре имя файла и нажать клавишу Enter - так будет создан пустой файл. При открытии файла запустится программа для работы с такими файлами. Этот способ может быть удобен, если пользователь находится в каталоге, где хотите создать новый документ. В этом случае он может оказаться быстрее, чем запуск программы, а затем выполнение диалога сохранения с поиском нужного места для сохранения файла.

Создать файл можно и через меню Проводника. Нужно зайти в меню «*Файл*», а через него в «*Создать*» и выбрать нужный тип документа, например, папку или архив. Такая опция доступна, если не выбирать другой объект в окне Проводника, а сразу перейти в меню через свободную область окна, находясь при этом в нужной директории.

Операция открытия файла или папки.

Открытие файла -это операция, в результате которой пользователь получает доступ к содержимому файла на своём мониторе. Если это текстовый файл, то будет показан его текст, если это фильм, то начнётся его воспроизведение, и так далее.

В зависимости от настроек операционной системы и используемых программ, существует несколько способов открыть файл или папку. Самый простой -дважды щёлкнуть по файлу или папке левой кнопкой мыши (ЛКМ). Однако, начинающим пользователям бывает непросто это сделать. В таком случае можно действовать следующим образом:

1. Щёлкнуть по файлу или папке один раз левой кнопкой мыши: он выделится.

2. Нажать клавишу Enter или кликнуть по нему правой кнопкой мыши (ПКМ).

3. Выбрать в контекстном меню пункт «*Открыть*».

Меню проводника Windows позволяет выполнять операции с файлами и папками и настраивать сам проводник. Чтобы открыть меню проводника, нажмите F10 на клавиатуре. Затем щёлкните левой кнопкой мыши по нужному файлу и выберите в меню пункт «*Файл*» ⇒ «*Открыть*».

Иногда Windows сама не знает, с помощью какой программы нужно открыть файл. Или пользователь хочет открыть этот файл в другой программе. Для этого необходимо щёлкнуть правой кнопкой мыши по файлу, выбрать в контекстном меню «*Открыть с помощью*» и указать нужную программу.

Переименование файлов и папок.

Имена всех объектов на жёстком диске компьютера уникальны в пределах одного каталога. Эти имена можно изменять сколько угодно раз, однако необходимо соблюдать требования файловой системы к именам. В случае ввода недопустимого имени компьютер выдаст предупреждение.

Для переименования объекта необходимо выделить его, щёлкнув один раз левой кнопкой мыши (ЛКМ), чтобы он подсветился цветом, а затем, после небольшой задержки, щёлкнуть ЛКМ ещё раз. Имя должно быть выделено синим цветом. Теперь можно ввести новое имя с клавиатуры и щёлкнуть левой кнопкой мыши в свободном месте окна Проводника или нажать клавишу Enter на клавиатуре.

Файлы и папки также можно переименовать с помощью всплывающего контекстного меню Проводника. Для этого щёлкните правой кнопкой мыши (ПКМ) на объекте и выберите пункт «*Переименовать*». Дальнейшие действия такие же, как описано выше.

Ещё один способ переименовать объект -использовать меню Проводника: выделите файл или папку щелчком ЛКМ и в меню Проводника выберите «*Файл*» → «*Переименовать*», а дальше действуйте как в других способах.

Операции копирования файлов и папок.

Чтобы не потерять важные файлы, рекомендуется создавать их копии. Копии могут также пригодиться для сравнения или использования в различных целях. Операция копирования файла выполняется так же, как и его перемещение, но с одним отличием: в меню нужно выбрать пункт «*Копировать*» или использовать комбинацию клавиш **Ctrl+C**. В результате в папке назначения будет создана копия исходного файла.

Вставить копию файла можно в ту же папку, что и оригинал, но при этом файлу необходимо присвоить новое имя. Также вставить файл можно и в другую папку по необходимости. Если же файл просто переносится, а не копируется, то для этого его нужно перетащить, удерживая при этом клавишу Ctrl.

Операция удаления папок и файлов.

Операционная система Windows предоставляет несколько способов удаления ненужных данных:

1. Первый способ заключается в том, чтобы выделить объект щелчком левой кнопкой мыши (ЛКМ), нажать клавишу Delete (Del) и подтвердить действие, нажав в появившемся окне кнопку «*Да*» или клавишу Enter на клавиатуре.

2. Второй способ удаления данных -через контекстное меню. Достаточно щёлкнуть на объекте правой кнопкой мыши (ПКМ), выбрать пункт «*Удалить*» и подтвердить свои намерения.

3. Третий способ -удалить данные через меню Проводника. Для этого нужно выделить объект левой кнопкой мыши и в меню Проводника выбрать меню «*Файл*» → «*Удалить*», после чего подтвердить действие.

Ещё один способ удалить данные -метод «*Перетащи и отпусти*» (или drag-and-drop). Нужно щёлкнуть левой кнопкой мыши по объекту и, удерживая её, переместить этот объект на иконку «*Корзины*», расположенной на рабочем столе.

Следует помнить, что при удалении папки все её содержимое, включая подпапки, также будет удалено.

Все удалённые данные оказываются в «Корзине» -специальной выделенной области на жестком диске компьютера, предназначенной для временного хранения удаленных данных перед их полным удалением из системы. Этот раздел можно открыть, щёлкнув на иконку «Корзина», расположенную на рабочем столе компьютера. Если случайно удалить файл или папку, эти данные можно восстановить в корзине. Нужно найти нужный объект в разделе и щёлкнуть по нему правой кнопкой мыши, чтобы выбрать в контекстном меню пункт «Восстановить».

В проводнике Windows можно откатить последние действия, если они по ошибке были выполнены. Для этого необходимо выбрать пункт меню «*Правка*» → «*Отменить*» или комбинацию клавиш Ctrl+Z.

Чтобы при удалении файлы не копировались в «Корзину», нужно удерживать клавишу Shift во время операции удаления. Объекты будут удалены окончательно и их больше не получится восстановить. Будьте осторожны с этим способом.

Групповые операции с файлами и папками.

Для облегчения работы с большим количеством объектов в операционной системе можно использовать функцию выделения группы объектов, что позволяет ускорить работу с файлами и папками.

Произвольная группа файлов и папок выделяется щелчком левой кнопкой мыши по объектам с нажатой клавишей Ctrl. Выделенные объекты подсвечиваются цветом.

Если нужно выделить группу файлов или папок, которые идут подряд, то сначала щёлкаем курсором левой кнопкой мыши на первый нужный объект, а затем удерживая при этом клавишу Shift, выделяем последний объект.

Выделить все объекты в текущем каталоге можно следующим образом: с помощью меню Проводника «Правка» ⇒ «Выделить всё»; комбинацией клавиш Ctrl+A.

Чтобы выделить все файлы и папки за исключением некоторых, можно поступить несколькими способами:

1. Сделать копирование всех нужных файлов, нажав Ctrl+C. Такой способ может занять довольно долгое время, если объектов много.

2. Выделить все объекты (Ctrl+A), а затем снять выделение с ненужных файлов и папок, щёлкнув по ним с нажатой Ctrl. Также можно обратить выделение, выбрав соответствующий пункт в меню «Правка». Оба способа займут меньше времени.

Для выделения группы файлов ещё используют метод выделения перемещением: зажимаем левую кнопку мыши и перемещаем курсор в нужное место окна. Указатель принимает форму рамки, которая ограничивает область выделения -отпускаем левую кнопку, чтобы завершить выделение. Можно отредактировать получившееся выделение, удерживая клавишу Ctrl.

Снять выделение можно любым способом: необходимо щёлкнуть мышью в свободном от выделенных объектов места. Дальнейшие операции с выделением повторят описанные ранее способы для одиночных объектов.

4.2 Базы данных

Термин «**База данных**» (БД) связан с такими понятиями, как «данные» и «информация». Более подробно эти понятия раскрыты в главе 1.

Хранящиеся в информационной системе данные должны быть доступны в необходимом виде для конкретной деятельности человека. Сегодня мы можем встретить как традиционный метод обработки данных с

ручным заполнением форм и помещением их в скоросшиватель, так и современные системы с электронными системами обработки информации на компьютерах. Несмотря на кардинальное различие, обе системы обязаны предоставить достоверную информацию в требуемое время, место и со скромными затратами ресурсов.

Информация получается из данных в процессе решения какой-либо задачи. Однако не вся информация может быть напрямую выведена из данных - например, человеческие эмоции, отражённые в искусстве, можно трактовать по-разному. А различные люди могут совершенно по-своему понимать даже простую совокупность цифр в качестве даты.

Для того чтобы извлечь информацию из каких-либо данных, необходимы определённые договорённости, которые называются интерпретацией данных. Также важно отметить, что нет универсального компактного кода, который мог бы охватить любую информацию, зачастую разнородную.

В контексте технических информационных систем информация, о которой пойдёт речь далее, - это информация, которую можно перевести на язык, понятный компьютеру. Кроме того, выбор компьютера как информационной системы обосновывается стоимостью хранения информации: в настоящее время она минимальна.

Часто на язык компьютера переводится только часть информации о какой-либо сфере деятельности человека. Например, это может быть телефонный справочник, перечень товаров, адреса клиентов фирмы и другие данные. Всё это называют базой данных (БД).

Также базой данных можно назвать совокупность связанной информации, объединённой общим признаком. Классическое определение базы данных по Дж. Мартину - «совокупность взаимосвязанных данных при такой их минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений в определённой предметной области человеческой деятельности». Предметной областью можно назвать фрагмент действительности, вызывающий интерес для исследования. Получается, база данных -это динамическая информационная модель некоторой области деятельности человека.

Чаще всего под базой данных понимают не просто набор связанных данных, а набор с программой обслуживания, которая обеспечивает взаимодействие пользователя с данными и возможность проводить операции. Сюда относится поддержка структуры данных, обеспечение целостности данных, добавление, удаление, поиск, сортировка и

фильтрация данных. По крайней мере минимальный набор таких операций осуществляет ядро (процессор) базы данных. Соответственно, БД состоит из набора данных, ядра базы данных и программы, реализующей интерфейс с пользователем.

Разнообразные операции с данными предоставляют Системы Управления Базами Данных (СУБД). Их можно назвать совокупностью языковых и программных средств, которые позволяют создать и поддерживать базу данных.

Помимо инструментов для поддержки организации данных и работы с ними, система управления базами данных также предоставляет пользователям развитый интерфейс, инструменты программирования высокого уровня, функции администрирования, а также средства защиты и обеспечения конфиденциальности информации. Таким образом, база данных представляет собой совокупность взаимосвязанных данных, в то время как СУБД является программной средой, способной управлять множеством баз данных и предоставляющей инструменты для их разработки.

Однако не каждый набор данных может быть классифицирован как база данных. Существуют определённые требования, которые необходимо выполнить, чтобы набор данных можно было считать базой данных:

1. Связность и логическая упорядоченность: Набор неупорядоченных данных не может считаться базой данных. Логическая упорядоченность подразумевает хранение только однородных данных. Например, поле «фамилия» должно содержать только фамилии, а не имена или отчества. Кроме того, эта информация не должна смешиваться с названиями предприятий.

2. Контролируемая избыточность данных: Каждый объект должен быть описан только один раз. Избыточность данных необходима для корректной работы системы. Некоторые дополнительные данные могут повысить эффективность и скорость поиска информации, а также облегчить восстановление данных после сбоев.

3. Целостность данных: Данные должны быть достоверными в любой момент времени.

4. Физическая и логическая независимость: Приложения должны быть независимы от изменений в логической структуре данных. Добавление или удаление некоторых характеристик описываемых объектов и связей между ними не должно требовать значительных переделок базы данных. На физическом уровне, данные должны быть независимыми от их размещения, типа носителя, организации и способа доступа к ним.

5. Безопасность и секретность: Защита данных от случайного или преднамеренного доступа неавторизованных лиц, а также присваивание определённых прав на изменение или использование данных являются аспектами безопасности данных. Секретность определена как возможность отдельных лиц или организаций решать, когда, кому и какое количество информацию можно передать.

Помимо классических баз данных, которые в основном предназначены для хранения текстовой и числовой информации, существуют и другие направления использования баз данных: БД мультимедиа (*multimedia databases*); географические информационные системы (GIS), используемые для хранения карт, спутниковых снимков и метеорологических данных; аналитическая обработка данных (*Data warehouses, Data mining и On-line analytical processing -OLAP*) - системы, применяющиеся для выделения и анализа необходимой информации из очень больших баз данных с целью принятия решений. Системы реального времени и активные базы данных (*Real-time* и *active database technology*) используются для мониторинга проектирования и производства; а также интернет-базы данных, включающие серверные БД и поисковые системы.

Появление баз данных было вызвано невозможностью традиционной файловой системы хранения данных справиться с растущим потоком информации и её обработкой. В отличие от традиционной файловой структуры, база данных хранит данные упорядоченно и структурированно, исключая совместное хранение разнородной информации.

В текстовых файлах информация может располагаться произвольно, а в электронных таблицах данные упорядочены по строкам и столбцам, но также размещаются произвольно -человек сам решает в момент создания таблицы, как лучше расположить данные. В свою же очередь, структура данных в базе данных строго фиксирована и определяется стандартом используемой модели данных.

Базы данных отличаются от файловой системы тем, что они хранят не только данные, но и метаданные. Метаданные представляют собой информацию о структуре данных и хранятся в словаре (системном каталоге) отдельно от самих данных. Любая СУБД может работать с разными наборами данных, поскольку структура хранения данных доступна при чтении этих данных. В файловой системе способ хранения данных устанавливается каждой программой, которая отвечает за хранение и обработку данных, и структура данных является встроенной в программу доступа.

Важным различием между базами данных и файловыми системами является наличие расширенных возможностей поиска информации. В базах данных используется индексированное хранение информации, которое в сочетании с упорядоченностью данных обеспечивает значительное повышение скорости поиска. Благодаря этому СУБД снимают ограничения, присущие файловым системам: разделение и изоляция данных, дублирование данных, зависимость от данных, несовместимость файлов и программ доступа и другие.

В целом, базы данных решают проблемы традиционной файловой систем хранения, обеспечивают эффективное управление данными и предоставляют возможность выполнять более сложные операции над информацией, делая их незаменимыми в современных информационных системах.

Свойства баз данных:

1. **Независимость данных**. Независимость данных должна обеспечиваться на логическом и физическом уровнях. Логическая независимость предполагает независимость приложений от изменения логической структуры данных. Физический уровень независимости предполагает независимость данных от их конкретного размещения, типа, организации и способа доступа (то есть от физической модели). Независимость данных на этом уровне обеспечивается развязкой первого и второго уровней. Это значит, что при изменении физической модели не потребуется производить изменения на концептуальном и внешнем уровнях.

В настоящее время этот уровень независимости обеспечивается СУБД и операционной системой, в отличие от ранних пор развития баз данных, хотя тогда он представлял собой большую проблему. Накопители на магнитных лентах были основным способом хранения данных, но единые стандарты размещения данных и управления накопителями отсутствовали. Поэтому при смене типа носителя приходилось полностью переписывать все программы доступа к данным.

2. **Избыточность данных**. Избыточность данных в первую очередь связана с хранением повторяющейся информации. Наличие такой информации не только приводит к увеличению размера базы данных, но и может привести к появлению несогласованного состояния данных.

Рассмотрим пример. Если необходимо хранить данные о покупках, сделанных в кредит, то для каждой покупки отдельного покупателя нужно повторно вводить его персональные данные. Например, ФИО и адрес. При

покупке одинаковых товаров разными покупателями необходимо каждый раз дублировать описание товара. Это приводит к таким проблемам:

- Лишние усилия. Приходится тратить время на ввод повторяющихся данных.
- Вероятность ошибки. Существует больший риск ошибки при вводе данных, которые уже есть в базе.
- Сложность обновления. Необходимо корректировать все записи при изменении сведений об объекте.
- Согласованность данных. Она может быть нарушена из-за наличия разных копий одних данных или при обновлении не всей информации, а также при непреднамеренном удалении информации, которая напрямую не связана с удаляемыми данными (аномалия удаления); или из-за добавления неполных данных (аномалии добавления).
- Скорость работы. Из-за избыточности данных время выполнения запросов, поиска и других операций снижается.

Чтобы избежать таких проблем, необходимо сохранять каждую логическую единицу данных только один раз. Избыточные данные должны присутствовать в системе только в том объёме, который необходим для нормального функционирования. Кроме того, некоторая избыточность требуется и для эффективности работы, ускорения поиска и восстановления после сбоев. Хранение согласованных данных является основой одного из свойств базы данных.

Обеспечение целостности базы данных является необходимым условием для её успешного функционирования. Целостность означает, что информация в базе данных является полной, непротиворечивой, согласованной и адекватно отражающей предметную область.

3. **Целостность и корректность данных**. Как правило, мы доверяем данным, которые мы видим в печатных изданиях. Во время подготовки книги редакторы проверяют информацию, чтобы сделать её правильной и соответствующей нормам. При работе с компьютером мы также стремимся к тому, чтобы информация была достоверной. Однако не все осознают, что ошибки или недочёты в данных могут привести к искажению результатов. Также нельзя полагаться исключительно на то, что компьютер не ошибается. Человеческий фактор всегда может внести искажения при нарочном или ненамеренном изменении информации.

В случае обнаружения ошибок, противоречий или некорректного изменения информации, необходимо принять меры для восстановления целостности базы данных.

Для поддержания целостности применяется система ограничений (constraints), которые представляют собой условия, которым должны соответствовать данные в базе. Ограничители целостности бывают трёх видов:

•Ограничители значений. К этой группе относятся правила, которые позволяют вводить только определённые данные, например, по заранее заданному типу и формату или в рамках установленного диапазона значений.

•Ссылочная целостность. Этот вид ограничений обеспечивает контроль соответствия между связанными данными и позволяет осуществлять каскадное удаление и обновление записей.

•Целостность записи. Целостность записи обеспечивается проверкой уникальности некоторых данных и обязателен ввод определённых данных.

Другим важным механизмом поддержания целостности является введение транзакций. Транзакция представляет собой последовательность операций над данными в базе, которая отслеживается от начала до завершения. Если транзакция не была успешно завершена, то производится отмена всех изменений, сделанных в рамках этой транзакции. Транзакции обладают четырьмя основными свойствами:

- Атомарность - в транзакции выполняются либо все операции, либо ни одна из них.
- Согласованность - каждая транзакция переводит базу данных из одного согласованного состояния в другое.
- Изолированность - транзакции выполняют независимо друг от друга.
- Безопасность - даже если работа с базой данных внезапно прерывается, данные сохраняются в полном объёме без потери информации.

Контроль транзакций особенно значим в многопользовательских базах данных, где транзакции могут выполняться параллельно. Компьютер не может обрабатывать параллельно выполняемые процессы, поэтому обычно процессы разбивают на сравнительно небольшие части и выполняют их поочерёдно. Когда две или более транзакции читают или модифицируют разные данные, проблем не возникает. Проблемы могут появиться, если транзакции обращаются к одним и тем же данным, поскольку от порядка выполнения частей транзакций будет зависеть конечное состояние данных.

В этом случае говорят о сериализации транзакций, то есть о составлении такого плана их выполнения (сериального плана), при котором суммарный эффект реализации транзакций такой же, как при их последовательном выполнении. Важно учитывать, что при параллельном

выполнении транзакций возможны конфликты, или блокировки. При возникновении конфликтов обычно производится откат одной или нескольких транзакций.

4. **Безопасность**. База данных - это важный ресурс, который нуждается в надёжной защите от различных опасностей: похищения и фальсификации данных, утраты конфиденциальности и целостности, потери доступности и непредумышленного или умышленного повреждения данных.

Для защиты от этих угроз существуют разнообразные контрмеры, включая наблюдение, защиту и восстановление с помощью компьютерных систем, а также правовые и административные процедуры.

Безопасность обеспечивается защитой объектов БД и программного кода от модификации. Запрещается редактирование по умолчанию, поддерживается блокировка, уровни изолированности транзакций и средства восстановления информации после сбоев, такие как создание контрольных точек, ведение журнала, протоколирование и создание архивных копий.

Секретность достигается шифрованием программ и данных, защитой с помощью пароля, авторизации и аутентификации пользователей, а также различными уровнями доступа к базе данных и её отдельным элементам, таким как таблицы, формы, отчёты и т.п.

4.2.1 Реляционные СУБД (SQL)

Реляционная модель данных (РМД) была разработана на основе теории отношений, которую создали два логика - американец *Чарльз Содерс Пирс* (1839–1914) и немец *Эрнст Шредер* (1841-1902).

Руководства по теории отношений содержат информацию о том, что множество отношений замкнуто относительно некоторых специальных операций. Это свойства отношений и легло в основу идеи РМД. Данная особенность позволила создать в реляционной модели язык манипулирования данными, связанный с исходной алгеброй.

Алгебра - это множество объектов с определённой совокупностью операций, которые замкнуты относительно основного множества. Американский математик *Эдгар Франк Кодд* в 1970 году сформулировал основные понятия и ограничения реляционной модели. Предложения Кодда оказались настолько эффективными для систем баз данных, что он был удостоен премии Тьюринга за разработку теоретических основ вычислительной техники.

Франк Кодд определил 13 правил реляционной модели, причём счёт начал с 0:

0. Реляционная система управления базами данных должна иметь возможность полностью управлять базой данных через её реляционные возможности.

1. В реляционных базах данных вся информация, включая имена таблиц и столбцов, должна определяться строго как табличные значения.

2. Любое значение в реляционной базе данных должно быть гарантированно доступно для использования через комбинацию имени таблицы, значения первичного ключа и имени столбца.

3. СУБД должна уметь работать с пустыми значениями (неизвестными или неиспользуемыми значениями), в отличие от значений по умолчанию и независимо для любых доменов.

4. Описание базы данных и её содержания должны быть представлены на логическом уровне в виде таблиц, которые можно использовать при запросах с помощью языка базы данных.

5. По крайней мере один из поддерживаемых языков должен иметь чётко определённый синтаксис и быть всеобъемлющим. Он должен поддерживать описание структуры данных и манипулирование ими, правила целостности, авторизацию и транзакции.

6. Все представления, которые теоретически можно обновлять, должны обновляться через систему.

7. СУБД поддерживает не только запрос на отбор данных, но и вставку, обновление и удаление данных.

8. На программы-приложения и специальные программы логически не влияют изменения физических методов доступа к данным и структур хранилищ данных.

9. На программы-приложения и программы логически не влияет изменение структур таблиц (в пределах разумного).

10. Язык базы данных должен быть способен определять правила целостности. Эти правила должны содержаться в онлайновом справочнике, и при этом не должно существовать способа их обойти.

11. На программы-приложения и специализированные программы не влияет первый или повторный раз использования данных.

12. С помощью языков низкого уровня невозможно обойти правила целостности, определённые с помощью языка баз данных.

Реляционная алгебра позволяет определить набор операций, которые могут быть использованы для запроса системе о том, как сформировать необходимое отношение в базе данных на основе определённых отношений.

Реляционные операторы замкнуты относительно понятия отношения, что означает, что выражения реляционной алгебры определяются над отношениями реляционных баз данных и результатом вычислений также являются отношения. Это свойство позволяет образовывать реляционные выражения, в которых вместо отношения - операнда некоторой реляционной операции находится вложенное реляционное выражение.

Выражения реляционной алгебры строятся на основе алгебраических операций высокого уровня. Их можно интерпретировать процедурно так же, как интерпретируются арифметические и логические выражения. Таким образом, запрос, выраженный на языке реляционной алгебры, может быть вычислен на основе вычисления элементарных алгебраических операций с учётом их старшинства и возможного наличия скобок.

Набор основных алгебраических операций реляционной алгебры состоит из восьми операций, которые делятся на два класса: **теоретико-множественные** и **специальные** реляционные операции. В состав теоретико-множественных операций входят традиционные операции над множествами (объединение, пересечение, разность и декартово произведение). Специальные реляционные операции включают выборку, проекцию, условное соединение и деление.

Теоретико-множественные операции.

Объединение. Результат объединения двух множеств (отношений) A и B - это новое множество (отношение) C, которое включает в себя элементы, присутствующие либо в множестве A, либо в множестве B, исключая дубликаты, возникающие из-за того, что некоторые элементы принадлежат обоим множествам.

Пусть заданы два множества $R_1=\{r_1\}$, $R_2=\{r_2\}$, где r_1 и r_2 - соответственно кортежи множеств R_1 и R_2, то их объединение описывается следующим образом:

$$R_1 \cup R_2 = \{\, r \mid r \in R_1 \vee r \in R_2 \,\} \qquad (12)$$

где r - кортеж нового множества, $\vee$ - операция логического сложения «логическое ИЛИ».

Рассмотрим пример применения операции объединения. Исходными множествами являются множества R_1 и R_2, которые содержат данные о коде поставщика и его местонахождении. Отношение R_3 содержит общий перечень данных поставщиков.

R_1	
Код поставщика	Город
К1	Москва
К2	Воронеж
К3	Казань
К4	Рязань
К5	Ярославль
К6	Энгельс

R_2	
Код поставщика	Город
К1	Москва
К7	Мурманск
К3	Казань
К2	Воронеж
К8	Кострома
К3	Казань

R_3	
Код поставщика	Город
К1	Москва
К2	Воронеж
К3	Казань
К4	Рязань
К5	Ярославль
К6	Энгельс
К7	Мурманск
К8	Кострома

Пересечение. Множество (отношение), которое состоит из кортежей, принадлежащих одновременно первому и второму множеству (отношениям), называют пересечением двух множеств (отношений) с эквивалентными схемами.

Пусть заданы два множества R_1 и R_2, содержащие кортежи r_1 и r_2, то их пересечение описывается выражением:

$$R_1 \bigcap R_2 = \{\ r \mid r \in R_1 \wedge r \in R_2\} \qquad (13)$$

где $\wedge$ - операция логического умножения («логическое И»).

Рассмотрим пример операции пересечения. Исходными данными являются отношения R_1 и R_2 из предыдущего примера. Результатом пересечения будет множество R_3, содержащее кортежи, принадлежащие одновременно и первому и второму множеству.

R_3	
Код поставщика	Город
К1	Москва
К2	Воронеж
К3	Казань

Разность. Множество (отношение), которое состоит из кортежей, принадлежащих первому множеству (отношению) и не принадлежащих второму множеству (отношению), называется разностью двух множеств

(отношений) с эквивалентными схемами. Пусть заданы два множества (отношения) R_1 и R_2, содержащие кортежи r_1 и r_2, то их пересечение описывается выражением:

$$R_1 / R_2 = \{\ r \mid r \in R_1 \wedge r \notin R_2\} \tag{14}$$

Рассмотрим пример операции на основе данных множеств (отношений) R_1 и R_2. Результатом разности R_1 - R_2 будет множество (отношение) R_3, содержащее кортежи, принадлежащие первому множеству и не принадлежащее второму множеству.

R_3	
Код поставщика	Город
К4	Рязань
К5	Ярославль
К6	Энгельс

Результатом разности $R_2 - R_1$ будет множество (отношение) R3, содержащее кортежи, принадлежащие второму множеству и не принадлежащее первому множеству (отношению).

R_3	
Код поставщика	Город
К7	Мурманск
К8	Кострома

Декартово произведение. Произведением множества (отношения) R_1 и множества (отношения) R_2 называется множество (отношение), кортежи которого получаются путём соединения (сцепления) каждого кортежа множества (отношения) R1 с каждым кортежем множества (отношения) R2. Пусть заданы два множества R_1 и R_2, содержащие кортежи r и q, то их декартово пересечение описывается выражением:

$$R_1 \otimes R_2 = \{(r,q) \mid r \in R_1 \wedge q \in R_2\} \tag{15}$$

Рассмотрим пример декартово произведения.

R_1	
Номер группы	Фамилия
23-1Б	Иванова
23-1М	Семенова
22-2Б	Кузнецова

R_2	
Профиль	Форма обучения
МКМТ	Очная
ГМУ	Заочная

R_3			
Номер группы	Фамилия	Профиль	Форма обучения
23-1Б	Иванова	МКМТ	Очная
23-1Б	Смирнова	ГМУ	Заочная
23-1М	Семенова	МКМТ	Очная
23-1М	Воронцова	ГМУ	Заочная
22-2Б	Кузнецова	МКМТ	Очная
22-2Б	Ткачев	ГМУ	Заочная

Специальные реляционные операции.

Фильтрация (выборка). Множество (отношение) R[a], которое включает те кортежи из исходного множества (отношения), для которых истинно условие выбора или фильтрации, называется результатом операции выбора или фильтрации, заданной на множестве (отношении) R в виде булевского выражения, определенного на атрибутах множества (отношения) R.

$$R[\alpha(r)]=\{r \mid r \in R \wedge \alpha(r)="Истина"\} \quad (16)$$

Условие $\alpha(r)$ может содержать термы сравнения (=, <>, <=,> =, <,>), константы, логические связки, скобки. Рассмотрим пример операции выборки. Необходимо выбрать из R_3 учащихся из номера группы «23-1М».

R_3			
Номер группы	Фамилия	Профиль	Форма обучения
23-1Б	Иванова	МКМТ	Очная
23-1Б	Иванова	ГМУ	Заочная
23-1М	Семенова	МКМТ	Очная
23-1М	Семенова	ГМУ	Заочная
22-2Б	Кузнецова	МКМТ	Очная
22-2Б	Кузнецова	ГМУ	Заочная

Тогда выражение для операции: $R_4=R_3$[Номер группы = «23-1М»]. Результат выборки:

R_4			
Номер группы	Фамилия	Профиль	Форма обучения
23-1М	Семенова	МКМТ	Очная
23-1М	Воронцова	ГМУ	Заочная

Проекция. Проекцией отношения R (A_1, A_2, ..., A_n) по подмножеству его атрибутов B = (A_{i1}, A_{i2}, ..., A_{ik}), которое является частью (A_1, A_2, ..., A_n), называют отношение со схемой, соответствующей набору атрибутов B. Оно содержит кортежи, которые получаются из кортежей исходного отношения R путём удаления значений, не принадлежащих атрибутам из B.

$$R[B] = \{\ r[B]\ \} \tag{17}$$

S_R= (A_1, A_2, ..., A_n), B = (A_{i1}, A_{i2}, ... , A_{ik}) $\subseteq S_R$,

Операция проектирования, которая также называется операцией вертикального выбора, позволяет получить только необходимые характеристики моделируемого объекта. Как правило, эта операция используется в качестве промежуточного шага в операциях горизонтального выбора или фильтрации. Кроме того, она применяется самостоятельно на заключительном этапе получения ответа на запрос.

Например, необходимо выбрать все фамилии, которые обучаются по профилю «ГМУ». Для этого необходимо из отношения R_3 выбрать фамилии с заданным названием, а потом полученное отношение спроектировать на столбец «Фамилия». Результатом выполнения этих операций будет отношение R_5:

R_4=R_3[Профиль = «ГМУ»], R_5=R_4[Фамилия]

R_3			
Номер группы	Фамилия	Профиль	Форма обучения
23-1Б	Иванова	МКМТ	Очная
23-1Б	Смирнова	ГМУ	Заочная
23-1М	Семенова	МКМТ	Очная
23-1М	Воронцова	ГМУ	Заочная
22-2Б	Кузнецова	МКМТ	Очная
22-2Б	Ткачев	ГМУ	Заочная

R_4			
Номер группы	Фамилия	Профиль	Форма обучения
23-1Б	Смирнова	ГМУ	Заочная
23-1М	Воронцова	ГМУ	Заочная
22-2Б	Ткачев	ГМУ	Очно-заочная

R_5
Фамилия
Смирнова
Воронцова
Ткачев

Условное соединение. Пусть R={r}, Q={q} - исходные отношения., S_R, S_Q - схемы отношений R и Q соответственно. При этом полагаем что заданы наборы атрибутов A и B:

$$A \subseteq \{A_i\}_{i=1,k};\ B \subseteq \{B_j\}_{j=1,m}; \tag{18}$$

и эти наборы состоят из ϑ-сравнимых атрибутов.

Тогда соединением отношений R и Q при условии β будет подмножество декартова произведения отношений R и Q, в котором содержатся такие кортежи, которые удовлетворяют условию β, причём это условие рассматривается как одновременное выполнение условий.

$A_i \ \theta_i \ B_i : i = 1,k,$

где k *– число атрибутов, входящих в наборы* A *и* B,

а θ_i *– конкретная операция сравнения.*

$A_i \ \theta_i \ B_i \ D;$

θ *– i – й предикат сравнения, определяемый из множества*

допустимых на домене D_i *операций сравнения.*

$$R[\beta]Q = \{(r,q) \mid (r,q) \mid r.A\,\theta_i\ q.B_i = \text{"Истина"}, i = 1,k\}. \tag{19}$$

Рассмотрим пример выполнения операции условного соединения.

R	
Номер группы	Фамилия
23-1Б	Иванова
23-1Б	Семенова
22-2Б	Кузнецова

Q	
Номер группы	Форма обучения
23-1Б	Очная
23-1Б	Очно-заочная
21-3М	Заочная

S		
Номер группы	Фамилия	Форма обучения
23-1Б	Иванова	Очная
23-1Б	Семенова	Очно-заочная
22-2Б	Кузнецова	Очно-заочная

Деление. Операция деления в некотором смысле является обратной операции произведения. Если отношение R_1 содержит атрибуты (A_1, A_2, ..., A_m, B_1, B_2, ..., B_n), а отношение R_2 - атрибуты (B_1, B_2. ..., B_n) и результирующее отношение R содержит атрибуты (A_1, A_2, ...,A_m), то кортеж

отношения R_1 включается в результирующее отношение, если его декартово произведение с отношением R_2 входит в R_1. Пример деления:

R_1	
Студент	Форма обучения
Иванова	Очное
Семенова	Очно-заочное
Кузнецова	Заочное
Фролова	Очное
Удальцова	Очно-заочное
Пирогова	Заочное
Маликова	Очное

R_2
Форма обучения
Очное

R_3
Студент
Иванова
Фролова
Маликова

Язык SQL

Язык SQL (*Structured Query Language*) был разработан в середине 1970-х годов сотрудниками компании IBM как высокоуровневый декларативный язык, основанный на реляционном исчислении. В предложениях SQL описывается то, какие данные необходимо получить, но не указывается, каким образом должен быть выполнен поиск результата.

Отличие предложений этого языка заключается в том, что они ориентированы главным образом на конечный результат обработки данных, а не на сам процесс этой обработки. Изначально язык назывался SEQUEL (Structured English QUEry Language), что говорит о намерении его создателей создать язык, близкий к естественному, и им это удалось. Команды SQL выглядят как обыкновенные предложения на английском языке, что делает процесс их изучения и понимания проще.

SQL возник как фирменный стандарт, но со временем приобрел статус международного языка. Его первая версия известна как стандарт ANSI SQL-89. Однако этот стандарт оказался далёк от совершенства, имел общий характер и допускал широкое толкование.

Благодаря опыту использования SQL, первый стандарт был дополнен новыми возможностями, необходимыми для реализации языка в составе коммерческих СУБД. На его основе разработали стандарт ANSI SQL-92, который считается истинно реляционным. После принятия этого стандарта можно было говорить о стандартной среде, SQL-ориентированной СУБД. И в настоящее время практически все современные реляционные СУБД поддерживают этот стандарт, хотя конкретные реализации, диалекты SQL его расширяют.

Типы данных языка SQL

Тип данных в языке структурированных запросов SQL — это атрибут, определяющий тип данных любого объекта, например колонки, переменной или выражения, который должен относиться к одному из типов данных SQL. В SQL есть шесть категорий типов данных, которые можно использовать в работе (таблица 15).

Таблица 15 – Точные типы числовых данных

Тип данных	**Возможные значения**	**Примечание**
Bit	0 или 1	Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.
Tinyint	От 0 до 255	Занимает 1 байт. Хорошо подходит для хранения небольших чисел.
Smallint	От -32 768 до 32 767	Занимает 2 байта
Int	От -2 147 483 648 до 2 147 483 647	Занимает 4 байта. Наиболее используемый тип для хранения чисел.
Bigint	От -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	Занимают в памяти 8 байт.
Decimal	Хранит числа с фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой. Данный тип может принимать два параметра precision и scale: DECIMAL(precision, scale). Параметр precision представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18. Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0.	
Money	Дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807	Представляет денежные величины и занимает 8 байт.
Smallmoney	Хранит дробные значения от -214 748.3648 до 214 748.3647	Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу DECIMAL(10,4).
Float	Хранит числа от -1.79E+308 до 1.79E+308	Занимает от 4 до 8 байт в зависимости от дробной части.
Real	От -340E+38 to 3.40E+38	Занимает 4 байта. Эквивалентен типу FLOAT(24).

Если нужно получить из базы данных числовое значение в строго указанном формате, необходимо выбрать один из точных числовых типов данных.

Таблица 16 – Примерные типы числовых данных

Тип данных	**Возможные значения**
float	От -1.79E + 308, до 1.79E + 308
real	От -3.40E + 38, до 3.40E + 38

Для записи чисел, которые невозможно точно представить в десятичном виде с ограниченным числом знаков (например, таких как одна треть или число пи), используют действительный (real) или плавающий (float) типы данных.

Данные действительного типа хранятся с точностью от 1 до 7 знаков, а в формате с плавающей точкой, который также называют форматом двойной точности, числа хранятся с 8-15 значащими цифрами. Действительный и плавающий типы данных применяют в научных приложениях для хранения чисел, не требующих точного двоичного выражения, при этом одна-две последние цифры могут не сохраняться при преобразованиях в двоичный формат.

Таблица 17 – Типы данных даты и времени

Тип данных	**Примечание**
datetime	От 1 января 1753, до 31 декабря, 9999
smalldatetime	От 1 января 1900, до 6 июня 2079
date	Сохраняет дату, как 30 июня 1991 года
time	Сохраняет время суток, как 12:30

Для дат в SQL и других системах используются два типа данных: datetime и smalldatetime. Тип smalldatetime охватывает период времени от 1 января 1900 года до 6 июня 2079 года и включает информацию о времени с точностью до минуты. Этого диапазона хватает для большинства проектов. Тип datetime можно использовать до 31 декабря 9999 года, что важно учитывать при решении проблемы 10К года.

Таблица 18 – Типы данных символьных строк

Тип данных	**Возможные значения**
char	Максимальная длина 8000 символов. (Фиксированная длина без Unicode символов)
varchar	Максимум 8000 символов. (Переменная длина данных не-Unicode).
text	Переменная длина данных, не Unicode с максимальной длиной 2147483647 символов.

Символьные данные, такие как имена или адреса, могут быть представлены либо с использованием типа данных фиксированной длины char, либо переменной длины varchar. Фиксированный размер предпочтителен, когда данные имеют одинаковую или сходную длину, например, при вводе индивидуального номера системы социальной безопасности, который часто используется в качестве идентификатора автора.

В большинстве случаев применение переменной длины для хранения данных не приводит к заметному увеличению времени обработки. Однако, если фамилия автора очень длинная, использование типа varchar становится оправданным, чтобы избежать потери значительного объёма памяти при использовании фиксированной длины поля. При выборе типа данных важно учитывать компромисс между потерей полезного объёма памяти при фиксированной длине и увеличением времени обработки при переменной длине данных.

Типы строк данных символов Unicode (таблица 19). **Юникод** (*англ. Unicode*) - стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира.

Таблица 19 – Типы данных символьных строк

Тип данных	**Возможные значения**
nchar	Максимальная длина 4000 символов. (Фиксированная длина Unicode)
nvarchar	Максимальная длина 4000 символов. (Переменная длина Unicode)
nvarchar(max)	Максимальная длина 231 символов (SQL Server 2005). (Переменная длина Unicode)
ntext	Максимальная длина 1,073,741,823 символа. (Переменная длина Unicode)

В базе данных двоичную информацию можно хранить в двух форматах: с фиксированной или переменной длиной. Данным фиксированной длины соответствует тип данных binary, а двоичным данным переменной длины - тип данных varbinary (таблица 20).

Таблица 20 – Двоичные типы данных

Тип данных	**Возможные значения**
binary	Максимальная длина 8000 байт (фиксированная длина двоичных данных)
varbinary	Максимальная длина 8000 байт. (Переменная длина двоичных данных)

varbinary(max)	Максимальная длина 231 байт (SQL Server 2005). (Переменная длина двоичных данных)
image	Максимальная длина 2147483647 байт. (Переменная длина двоичных данных)

В языке SQL есть два основных подмножества:

1. **SQL-DDL** (*Data Definition Language*) - он используется для определения структуры и обеспечения целостности баз данных. Команды этого подмножества позволяют создавать и удалять базы данных, а также управлять пользователями, создавать, изменять и удалять таблицы, последовательности и представления.

2. **SQL-DML** (*Data Manipulation Language*) - это подмножество предназначено для манипулирования данными. С помощью команд SQL-DML можно добавлять данные в базу, изменять их, извлекать и удалять по необходимости. Также он позволяет управлять транзакциями.

Общая структура языка SQL проиллюстрирована на рисунке 49. Кроме DDL и DML, язык SQL включает средства создания запросов, инструменты управления транзакциями, средства администрирования и так называемый программный SQL.

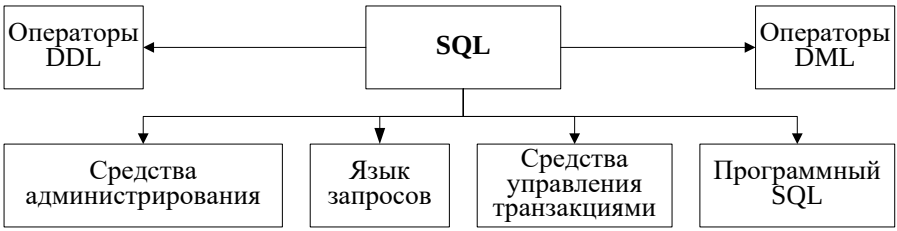

Рисунок 49. Структура SQL

В реляционной модели **таблица** (*Table*) является основным хранилищем информации в базе данных.

Представление (*View*) - это виртуальная (логическая) таблица, которая представляет собой результат поименованного запроса. В отличие от обычных таблиц реляционной базы данных, представление не является самостоятельной частью набора данных, хранящегося в базе. Содержимое представления динамически вычисляется на основе данных, находящихся в реальных таблицах.

Индекс (*Index*) - это объект базы данных, который создаётся для ускорения процесса поиска данных.

Таблица 21 – Операторы SQL-DDL

Оператор	**Выполняемая команда**
Create Database	Создать базу данных
Drop Database	Удалить базу данных
Create Table	Создать таблицу
Alter Table	Изменить таблицу
Drop Table	Удалить таблицу
Create Domain	Создать домен
Alter Domain	Изменить домен
Drop Domain	Удалить домен
Create Collation	Создать последовательность
Drop Collation	Удалить последовательность
Create View	Создать представление
Drop View	Удалить представление

Рассмотрим пример создания таблицы:

```
CREATE TABLE <ИМЯ ТАБЛИЦЫ>
 (<ИМЯ СТОЛБЦА> <ТИП СТОБЦА>,
  PRIMARY KEY (<ИМЯ СТОЛБЦА >),
  REFERENCES <ИМЯ ТАБЛИЦЫ> (<ИМЯ СТОЛБЦА>));
```

При создании таблицы необходимо указывать имя таблицы и список столбцов. Для каждого столбца обязательно указываются его имя и тип, а также дополнительно могут быть указаны параметры: `PRIMARY KEY` - столбец является первичным ключом. Конструкция `REFERNECES <ИМЯ ТАБЛИЦЫ> <ИМЯ СТОЛБЦА>` определяет, что данный столбец является внешним ключом и указывает на ключ какой таблицы он ссылается.

Пример удаления таблицы:

```
DROP TABLE <имя_таблицы>
```

Команда **ALTER TABLE**, используемая в SQL, позволяет добавлять, удалять или изменять столбцы в существующей таблице. Для выполнения этих действий пользователю необходимо иметь системную привилегию ALTER ANY TABLE или же таблица должна находиться в схеме пользователя. Рассмотрим синтаксис на примерах:

Добавление столбцов.

```
ALTER TABLE <имя_таблицы> ADD (<имя_столбца>
<тип_столбца> [NOT NULL] [UNIQUE | PRIMARY KEY]
[REFERENCES <имя_мастер_таблицы>[<имя_столбца>]],...)
```

Удаление столбцов.

```
ALTER TABLE <имя_таблицы> DROP (<имя_столбца>,...)
```

Модификация типа столбцов.

ALTER TABLE <имя_таблицы> **MODIFY** (<имя_столбца> <тип_столбца> [NOT NULL] [UNIQUE | PRIMARY KEY] [REFERENCES <имя_мастер_таблицы> <имя_столбца,...)

Примеры работы с индексами. Создание индекса:

CREATE [UNIQUE] **INDEX** <имя_индекса> ON <имя_таблицы> (<имя_столбца>,...)

Команда создаёт индекс с заданным именем для таблицы <имя_таблицы> по столбцам, которые входят в список, указанный в скобках. Этот индекс часто имеет структуру бинарного дерева (B-дерева), но могут применяться и другие структуры. Создание индексов существенно ускоряет работу с таблицами. Если дополнительно указан параметр UNIQUE, то система управления базами данных будет проверять каждое значение индекса на уникальность.

Пример удаления индекса:

DROP INDEX <имя_индекса>

Операторы DML (Data Manipulation Language) представляют собой операторы манипулирования данными - они используются для создания, изменения и извлечения данных из таблиц баз данных. С их помощью можно выполнять различные манипуляции с данными, такие как добавление новых записей, изменение существующих записей и удаление ненужных записей.

Таблица 22 - Режимы открытия файла

Оператор	Выполняемая команда
Select	Отобрать строки из таблиц
Insert	Добавить строки в таблицу
Update	Изменить строки в таблице
Delete	Удалить строки в таблице
Commit	Зафиксировать внесенные изменения
Rollback	Откатить внесенные изменения

Оператор SELECT. Команда SELECT служит для извлечения информации из базы данных. С её помощью можно выбрать одну или несколько строк и/или столбцов из одной или нескольких таблиц. Результатом выполнения SELECT всегда является таблица -даже если это всего одно число, оно всё равно рассматривается как таблица с одной строкой и одним столбцом.

Во время выполнения оператора SELECT данные, которые запрашиваются, обычно блокируются от изменений. Тем не менее, выполнение SELECT не изменяет данные, уже хранящиеся в базе данных.

С помощью SELECT можно искать и просматривать данные различными способами. Эти способы правильны, но могут существенно отличаться по времени исполнения -что особенно важно для больших баз данных. По этой причине написание запросов на языке SQL требует серьёзных усилий. Этому надо учиться, как учатся решать математические задачи или составлять алгоритмы решения задач на ЭВМ.

```
SELECT [ALL | DISTINCT] <список выбираемых полей | *>
FROM [<Имя базы данных.>] <Имя таблицы> | <Имя представления>]
[WHERE <Условия выборки или соединения>]
[GROUP BY <Выражение>]
[HAVING <Выражение>]
[ORDER BY <Выражение>]
```

В предложении SELECT команды SQL определяются поля (столбцы), которые будут включены в результат запроса. По умолчанию (ALL) выводятся все записи, которые удовлетворяют условиям запроса, что может привести к повторению строк. Чтобы избежать дублирования, можно использовать опцию DISTINCT, которая исключает повторяющиеся записи из результата запроса. Однако стоит учесть, что строки, где некоторые значения полей идентичны, а некоторые различаются, останутся в наборе.

При указании аргумента <список выбираемых полей> необходимо перечислить поля и допустимые выражения, включенные в результирующую таблицу. Каждый элемент списка становится отдельным столбцом при генерации результатов запроса. Можно включать имена выбираемых таблиц или их псевдонимы (при необходимости) с использованием символа «.» для разделения имени и псевдонима. Если указать символ «*» в аргументе, все столбцы всех таблиц будут выведены в результатах.

В предложении FROM перечисляются названия таблиц, с которых будет выполняться запрос. Для каждой таблицы можно добавить локальный псевдоним через один или несколько пробелов после названия таблицы.

Условие отбора записей из исходных таблиц задается в предложении **WHERE**.

Группировка строк в запросе выполняется с помощью предложения **GROUP BY**. Она основана на значении одного или нескольких полей результирующей таблицы.

Предложение **HAVING** фильтрует группы, чтобы включить их в результаты запроса. Его следует использовать только с предложением

GROUP BY. Именно HAVING позволяет задать условие фильтрации для групп.

Сортировка результатов запроса выполняется с применением предложения **ORDER BY**, основанного на одном или нескольких полях результирующей таблицы, и в порядке возрастания по умолчанию.

Оператор INSERT. Оператор INSERT позволяет добавлять новые записи в таблицу, при этом значения столбцов могут быть либо заданы как литеральные константы, либо получены в результате выполнения подзапроса. Если значения столбцов задаются как литеральные константы для каждой строки, то для вставки используется отдельный оператор INSERT, а если значения столбцов являются результатом выполнения подзапроса, то будет вставлено столько строк, сколько будет возвращено подзапросом.

```
INSERT INTO <имя таблицы> [(<имя столбца>,...)]
{VALUES (<значение столбца>,…)}
| <выражение запроса>
| {DEFAULT VALUES}
```

Оператор UPDATE. С помощью одного оператора UPDATE можно задать значения для любого количества столбцов, но только однократно для каждого столбца. Если оператор UPDATE применяется без предложения WHERE, то будут обновлены все строки таблицы.

Если столбец допускает NULL-значение, можно явно указать NULL. Также можно заменить текущее значение столбца значением по умолчанию (DEFAULT) для этого столбца.

```
UPDATE <имя таблицы>
SET {<имя столбца> = {<выражение для вычисления значения столбца>
| NULL
| DEFAULT},...}
[ {WHERE <предикат>}]
```

Оператор Delete. Оператор DELETE удаляет строки из временных или постоянных базовых таблиц, а также представлений или курсоров. В случае с представлениями и курсорами действие оператора распространяется на те базовые таблицы, из которых были извлечены данные для этих представлений или курсоров.

```
DELETE FROM <имя таблицы >
[WHERE <предикат>];
```

Если в запросе отсутствует предложение WHERE, то в результате будут удалены все строки из таблицы или представления, которое должно быть

обновляемым. В Transact-SQL эту операцию, удаление всех строк из таблицы, можно выполнить быстрее с помощью специальной команды.

Оператор COMMIT. Он позволяет успешно завершить транзакцию, все изменения, внесенные в базу данных, фиксируются. Успешное завершение программы, инициировавшей транзакцию, означает успешное завершение транзакции.

Оператор Rollback. Оператор позволяет отменить транзакцию, т.е. вернуть БД в состояние, в котором она находилась на момент начала транзакции. Ошибочное завершение программы прерывает транзакцию.

4.2.2 OLAP системы

В процессе анализа данных и поиска решений часто возникает необходимость в построении зависимостей между различными параметрами, и число таких параметров может варьироваться в широких пределах. Традиционные средства анализа, оперирующие данными в виде таблиц реляционной базы данных, не могут в полной мере удовлетворять таким требованиям. Основоположник реляционной модели БД *Э. Ф. Кодд* рассмотрел её недостатки ещё в 1993 году, указав на невозможность «объединять, просматривать и анализировать данные с точки зрения множественности измерений, наиболее понятным для аналитиков способом».

Измерение - это последовательность значений одного из анализируемых параметров. Например, для параметра «время» - это будет последовательность календарных дней, а для параметра «регион» - список городов. Множественность измерений предполагает представление данных в виде многомерной модели, где по измерениям откладывают параметры, относящиеся к анализируемой предметной области. Множественность измерений предполагает представление данных в виде многомерной модели. Параметры, имеющие отношения к предметной области, откладываются по измерениям в многомерной модели, что позволяет проводить анализ данных.

По определению Кодда, многомерное концептуальное представление (*multi-dimensional conceptual view*) - это множественная перспектива, состоящая из нескольких независимых измерений, с помощью которых можно анализировать определённые совокупности данных. Одновременный анализ по нескольким измерениям и называется **многомерным анализом**.

Каждое измерение может быть представлено в виде иерархической структуры. К примеру, иерархия измерения «Исполнитель» может выглядеть так: «*предприятие - подразделение - отдел - служащий*». Более того, некоторые измерения могут иметь несколько видов иерархического представления. Например, измерение «время» может включать две иерархии со следующими уровнями: «*год - квартал - месяц - день*» и «*неделя - день*».

На пересечениях осей измерений (*dimensions*) располагаются данные, которые количественно характеризуют анализируемые факты (*measures*). В число этих фактов могут входить объёмы продаж, выраженные в единицах продукции или в денежном выражении, остатки на складе, издержки и другое.

Многомерную модель данных можно представить в виде гиперкуба. Его рёбра - это измерения, а ячейки - меры (рисунок 50).

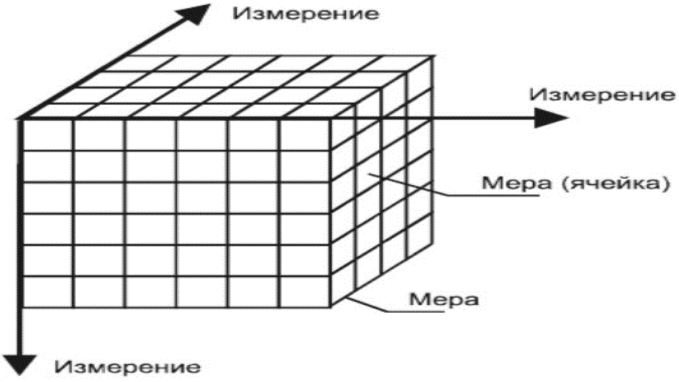

Рисунок 50. Представление данных в виде куба

Над гиперкубом можно выполнять различные операции, в числе которых срез (*Slice*) (рисунок 51).

Срез формирует подмножество многомерного массива данных, которое соответствует определённому значению одного или нескольких элементов измерений, не включённых в это подмножество.

Например, при выборе элемента «Факт» измерения «Сценарий» срез данных будет представлять собой подкуб, который объединяет все остальные измерения. Значения, которые не были включены в срез, связаны с теми элементами измерения «Сценарий», которые не определяли срез.

Для конечного пользователя термин «срез» чаще всего обозначает двумерную проекцию куба.

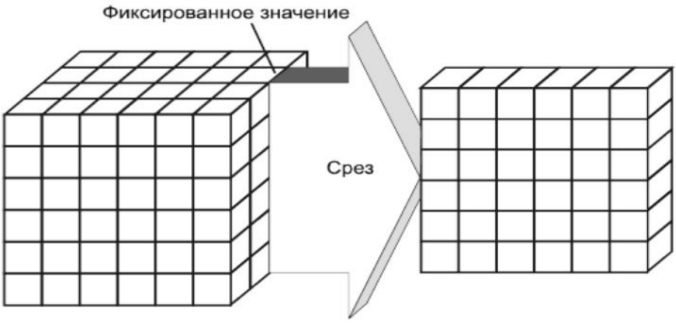

Рисунок 51. Операция срез

Вращение (*Rotate*) - это изменение расположения измерений, представленных в отчёте или на отображаемой странице. Эта операция позволяет переставить строки и столбцы таблицы или переместить интересующие измерения в колонках и строках отчёта для придания ему желаемого вида (рисунок 52).

В качестве примера можно привести отчёт, в котором элементы измерения «Время» располагаются поперёк экрана (то есть являются заголовками столбцов таблицы), а элементы измерения «Продукция» - вдоль экрана (заголовками строк таблицы). После применения операции вращения элементы измерения «Продукция» будут расположены горизонтально, а элементы измерения «Время» - вертикально.

Также примером может послужить преобразование отчёта с измерениями «Меры» и «Продукция», расположенными по вертикали, и измерением «Время», расположенным по горизонтали, в отчёт, у которого измерение «Меры» расположено вертикально, а «Время» и «Продукцию» можно увидеть по горизонтали. При этом элементы измерения «Времени» будут над элементами измерения «Продукции».

Описываемая нами операция может применяться также для преобразования отчёта, в котором измерение «Время» располагается горизонтально, а измерение «Продукция» - вертикально, в такой отчёт, где измерение «География» (или Pivot) представлено вертикально, а измерение «Время» - горизонтально.

Консолидация (*Drill Up*) и детализация (*Drill Down*) - это операции, которые позволяют переходить от детального (*down*) представления данных к агрегированному (*up*) и наоборот (рисунок 53). Направление детализации (обобщения) может быть задано по иерархии отдельных измерений или согласно другим отношениям, установленным в рамках измерений или между ними.

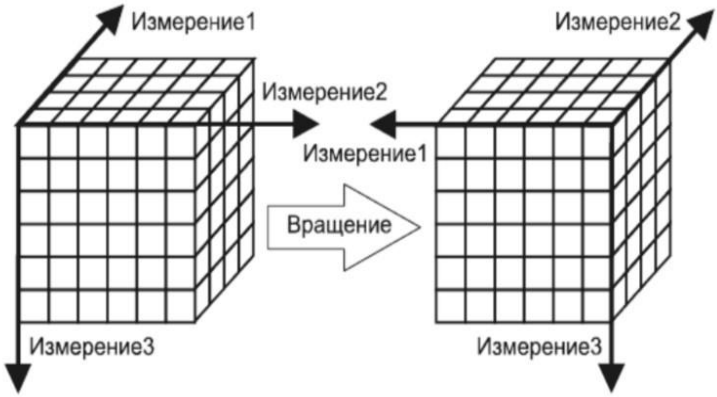

Рисунок 52. Операция вращение

Например, если при анализе данных об объёмах продаж в Северной Америке выполнить операцию Drill Down для измерения «Регион», то на экране появятся его элементы: «Канада», «Восточные штаты Америки» и «Западные штаты Америки». После дальнейшей детализации элемента «Канада» будут показаны его составляющие: «Торонто», «Ванкувер», «Монреаль» и т. д.

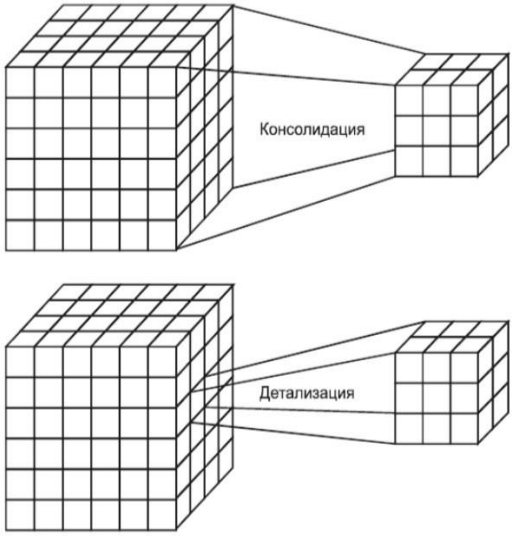

Рисунок 53. Операция вращение

С концепцией многомерного анализа данных тесно связан оперативный анализ, который выполняется с помощью OLAP-систем.

OLAP (On-Line Analytical Processing) - технология оперативной аналитической обработки данных, которая использует методы и средства сбора, хранения и анализа многомерных данных для поддержки процессов принятия решений. Основное назначение OLAP-систем - поддержка аналитической деятельности и предоставление ответов на произвольные (ad-hoc) запросы пользователей-аналитиков. Цель такого анализа - проверка возникающих гипотез.

Основоположником технологии OLAP считается Э. Кодд - основатель реляционного подхода. В 1993 году он опубликовал статью «OLAP для пользователей-аналитиков: каким он должен быть», в которой изложил основные концепции оперативной аналитической обработки и определил 12 требований, которым должны соответствовать продукты, выполняющие оперативную аналитическую обработку. Кодд определил 12 правил для OLAP (Online Analytical Processing), которые включают в себя следующие ключевые пункты:

1. **Многомерность**. На уровне концепции, данные должны быть представлены в виде многомерной модели, что делает их анализ и восприятие информации более простыми.

2. **Прозрачность**. OLAP-система скрывает от пользователя реальную реализацию многомерной модели и её компоненты. Пользователю предоставляется согласованная и целостная модель данных при любом расположении данных.

3. **Доступность**. Необходимо обеспечить пользователю доступ к данным, независимо от того, как и где они хранятся. В результате пользователь получает согласованную и целостную модель данных.

4. **Постоянная производительность отчетности**. Производительность систем OLAP не должна существенно ухудшаться при увеличении количества измерений.

5. **Клиент-серверная архитектура**. OLAP может работать в среде клиент-сервер, так как большинство данных, которые требуют оперативной аналитической обработки, хранятся распределенно. Компоненты серверного инструмента OLAP должны быть интеллектуальными и позволять объединять различные схемы баз данных для обеспечения прозрачности.

6. **Равноправие измерений**. Измерения в многомерной концептуальной модели должны быть равноправными. При необходимости, дополнительные атрибуты могут быть добавлены к измерениям.

7. **Динамическое управление разреженными матрицами**. Разработанные системы должны обеспечивать обработку разреженных матриц. Скорость доступа должна оставаться одинаковой вне зависимости от положения ячеек данных внутри модели.

8. **Поддержка многопользовательского режима**. Множество пользователей могут совместно работать с одной аналитической моделью или создавать разные модели из одних и тех же данных. Система обеспечивает безопасность и целостность данных при чтении и записи.

9. **Неограниченные перекрестные операции**. OLAP должна поддерживать функциональные отношения между ячейками гиперкуба при любых операциях среза, вращения, консолидации или детализации. Преобразование отношений должно выполняться системой автоматически.

10. **Интуитивная манипуляция данными**. Гиперкуб методов среза, детализации и консолидации должен быть реализуем без чрезмерного усилия пользователя. Измерения в аналитической модели обеспечивают необходимую информацию для манипуляций.

11. **Гибкие возможности получения отчетов**. Системы OLAP поддерживают разнообразные способы визуализации отчетов. Строки, столбцы и страницы отчета могут показывать от 0 до N измерений (где N - число измерений в аналитической модели). Каждое измерение позволяет показывать любое подмножество значений в любом порядке.

12. **Неограниченная размерность и число уровней агрегации**. Исследование о возможном числе необходимых измерений в аналитической модели показало, что одновременно может использоваться до 19 измерений, поэтому аналитический инструмент должен предоставлять хотя бы 15 измерений, а предпочтительно - 20. При этом нет ограничений по числу определяемых пользователем-аналитиком уровней агрегации и путей консолидации для каждого из общих измерений.

Набор требований, определяющих OLAP (online analytical processing - оперативную аналитическую обработку данных), часто вызывает нарекания. Дело в том, что некоторые из этих требований являются жёсткими правилами (правила 1, 2, 3, 6), а некоторые - неформализованными пожеланиями (правила 10, 11). Поэтому точно определить OLAP даже на основе знаменитого перечня требований Кодда (Codd) достаточно сложно. В 1995 г. Кодд добавил к этому списку ещё шесть правил:

13. **Пакетное извлечение против интерпретации**. OLAP-система (online analytical procession system - система оперативной аналитической

обработки данных) должна обеспечивать эффективный доступ как к собственным, так и к внешним данным.

14. **Поддержка всех моделей OLAP-анализа**. OLAP систма должна поддерживать все четыре модели анализа данных, определённые Коддом: категориальную, толкователую, умозрительную и стереотипную.

15. **Обработка ненормализованных данных**. OLAP система должна быть интегрирована с ненормализованными источниками данных. Изменения данных, выполненные в среде OLAP, не должны влиять на исходные внешние системы.

16. **Сохранение результатов OLAP**. Хранение их отдельно от исходных данных - OLAP-ситема должна после модификации исходных данных сохранять результаты отдельно. Иными словами, обеспечивается безопасность исходных данных.

17. **Исключение отсутствующих значений**. OLAP система, представляя данные пользователю, должна отбрасывать все отсутствующие значения. Это значит, что отсутствие данных должно отличаться от нулевых значений.

18. **Обработка отсутствующих значений**. Обработка отсутствующих значений в OLAP-системе должна происходить таким образом, чтобы система игнорировала все отсутствующие значения без учёта их источника, что связано с 17-м правилом работы с данными.

Кодд разбил все 18 правил на следующие четыре группы, назвав их особенностями. Эти группы получили названия B, S, R и D.

Основные особенности (B):

- многомерное концептуальное представление данных;
- интуитивное манипулирование данными;
- доступность;
- пакетное извлечение против интерпретации;
- поддержка всех моделей OLAP-анализа;
- архитектура «клиент-сервер»;
- прозрачность;
- многопользовательская поддержка.

Специальные особенности (S):

- обработка ненормализованных данных;
- сохранение результатов OLAP: хранение их отдельно от исходных данных;
- исключение отсутствующих значений;
- обработка отсутствующих значений.

Особенности представления отчётов (R):

- гибкость формирования отчётов;
- стандартная производительность отчётов;
- автоматическая настройка физического уровня.

Управление измерениями (D):

- универсальность измерений;
- неограниченное число измерений и уровней агрегации;
- неограниченные операции между размерностями.

Одной из наиболее известных спецификаций для OLAP является FASMI, которая была создана в 1995 году Найджелом Пендсом и Ричардом Критом на основе анализа правил Кодда.

F (Fast) - OLAP-система должна обеспечивать выдачу большинства ответов пользователям в пределах приблизительно 5 секунд. При этом самые простые запросы обрабатываются в течение 1 секунды, и очень немногие более 20 секунд. Недавнее исследование в Нидерландах показало, что конечные пользователи воспринимают процесс неудачным, если результаты не получены по истечении 30 секунд - они теряют мысль, при этом качество анализа страдает. Такой скорости нелегко достигнуть при обработке большого количества данных, особенно если требуются специальные вычисления «на лету». Для достижения данной цели используются разные методы, включая применение аппаратных платформ с большей производительностью.

Анализ (Analysis) - OLAP-система должна справляться с любым анализом данных и обеспечивать его сохранение в виде, доступном для конечного пользователя. Система должна позволять пользователю определять новые вычисления в процессе анализа и формировать отчеты, а все требуемые функциональные возможности должны быть понятны конечному пользователю.

Защита данных (Shared) - Система предоставляет доступ к данным только авторизованным пользователям, обеспечивая защиту информации на соответствующем уровне. Обработка модификаций данных пользователей должна выполняться своевременно и безопасным способом.

Многомерность (Multidimensional) - Концептуальное представление данных обеспечивает эффективное применение иерархий и множественных иерархий, а необходимое число измерений зависит от приложения. Кроме того, она также не определяет используемую технологию БД, но обеспечивает необходимую гибкость для обработки различных типов данных.

Функциональность для конечного пользователя (Information) - Мощность системы измеряется не объёмом хранимой информации, а

количеством входных данных, которые она может обработать. В этом смысле мощность OLAP-продуктов весьма различна: большие системы могут оперировать по крайней мере в 1000 раз большим количеством данных. Также следует учитывать множество факторов, включая дублирование данных, требуемую оперативную память, использование дискового пространства, эксплуатационные показатели, интеграцию с информационными хранилищами и так далее.

OLAP-система имеет два основных компонента:

OLAP-сервер - хранит данные, производит над ними операции и формирует многомерную модель данных на концептуальном уровне. Сейчас OLAP-серверы используются вместе с хранилищами данных (ХД) или витринами данных (ВД).

OLAP-клиент - обеспечивает пользователю удобный доступ к многомерной модели, чтобы проводить аналитику.

OLAP-серверы формируют гиперкуб, с которым пользователи выполняют все необходимые манипуляции для анализа информации. Слой сервера скрывает от пользователя способ реализации многомерной модели. Но этот способ важен, так как от него зависят характеристики OLAP-системы: производительность и требуемые ресурсы.

Реализация многомерной модели может быть осуществлена следующими способами (их также можно называть архитектурами OLAP):

- MOLAP (Multidimensional OLAP) - использует многомерные базы данных.
- ROLAP (Relational OLAP) - использует реляционные базы данных.
- HOLAP (Hybrid OLAP) - использует и многомерные, и реляционные базы.

Кроме того, часто в литературе по OLAP можно встретить аббревиатуры: DOLAP (Desktop OLAP) обозначает недорогой и простой в использовании OLAP, предназначенный для локального анализа и представления данных, загружаемых из реляционной или многомерной БД на машину клиента. JOLAP (Java OLAP API) представляет собой основанную на Java коллективную OLAP API-инициативу, предназначенную для создания и управления данными и метаданными на серверах OLAP. Её разработал Hyperion Solutions. В группу, определяющую предложенный API, входят IBM, Oracle и другие компании.

MOLAP. Для хранения и управления данными многомерные базы данных используют MOLAP-серверы, данные при этом хранятся в упорядоченных многомерных массивах, которые подразделяются на гиперкубы и поликубы. В гиперкубе все ячейки, которые хранятся в базе

данных, имеют одинаковую мерность, то есть находятся в максимальном базисе измерений. В поликубе каждая ячейка хранится с собственным набором измерений, все сложности обработки данных такого типа перекладываются на внутренние механизмы системы.

Данные, представленные в многомерном виде, физически хранятся в «плоских» файлах. Куб представляется в виде одной плоской таблицы, в которую построчно вписываются все возможные комбинации измерений с соответствующими им значениями измерений. Например, таблица 23.

Таблица 23 – Таблица MOLAP

Регион	**Продукт**	**Квартал**	**Продажи**
Р1	П1	1	6
Р2	П2	2	3
Р1	П2	1	9
Р2	П1	2	6

Среди инструментов MOLAP можно выделить следующие:

Essbase - инструменты корпорации Oracle, которые используют многомерную базу данных;

Express Server - веб-среда, функционирующая на основе Oracle database;

Yellowfin - инструменты бизнес-аналитики, позволяющие создавать отчёты и информационные панели;

Clear Analytics - бизнес-решение на базе Excel;

SAP Business Intelligence - решения для бизнес-аналитики от компании SAP.

ROLAP. Согласно мнению Кодда, реляционные базы данных были, есть и будут наиболее подходящей технологией для хранения информации. Существует потребность не в новой технологии баз данных, а скорее в средствах анализа, которые могли бы дополнить функции существующих систем управления базами данных, а также обладали бы достаточной гибкостью для обеспечения разных видов интеллектуального анализа, характерных для OLAP.

В настоящее время широко применяются две основные схемы реализации многомерного представления информации с применением реляционных таблиц: схема «звезда» (рисунок 54) и схема «снежинка» (рисунок 55).

Основными элементами схемы многомерной модели данных являются денормализованная таблица фактов Fact Table и множество таблиц измерений Dimension Tables. Таблица фактов включает информацию об объектах или событиях, которая будет подвергнута дальнейшему анализу. Обычно выделяют четыре типа фактов:

1. Факты, связанные с транзакциями (Transaction Facts), основаны на отдельных событиях (например, телефонный звонок или снятие денег со счёта с помощью банкомата).

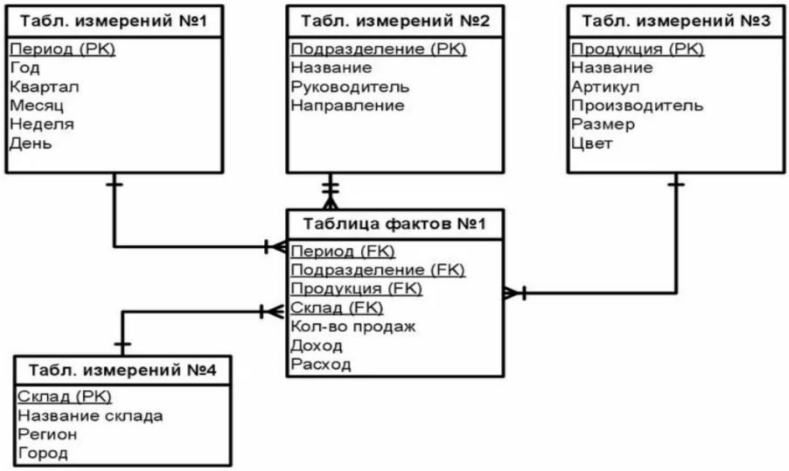

Рисунок 54. Схема «звезда»

2. Факты-«моментальные снимки» (Snapshot facts) основаны на состоянии объекта в определённые моменты времени (например, объём продаж за день или дневная выручка).

3. Факты, относящиеся к элементам документа (Line-item facts), связаны с определённым документом (например, счётом за товар или услуги) и содержат подробную информацию об элементах этого документа (например, количестве, цене, проценте скидки).

4. Факты о событиях и состояниях объекта (Event or state facts) представляют возникновение события без деталей о нём (например, просто факт продажи или факт отсутствия таковой без иных подробностей).

Таблица фактов обычно содержит уникальный составной ключ, который объединяет первичные ключи таблиц измерений. Ключевые и некоторые не ключевые поля должны соответствовать измерениям гиперкуба. Помимо этого, таблица фактов может содержать одно или несколько числовых полей, на основании которых в дальнейшем будут получены агрегатные данные.

Таблицы измерений содержат нечасто изменяемые данные и включают информации о членах нижних уровней иерархии соответствующих измерений. Таблицы измерений также содержат как минимум одно описательное поле (обычно с именем члена измерения) и, как правило,

целочисленное ключевое поле для однозначной идентификации члена измерения. Если измерение, соответствующее таблице, содержит иерархию, то таблица также может содержать поля, указывающие на «родителя» данного члена в этой иерархии. Скорость роста таблиц измерения должна быть незначительной по сравнению со скоростью роста таблицы фактов.

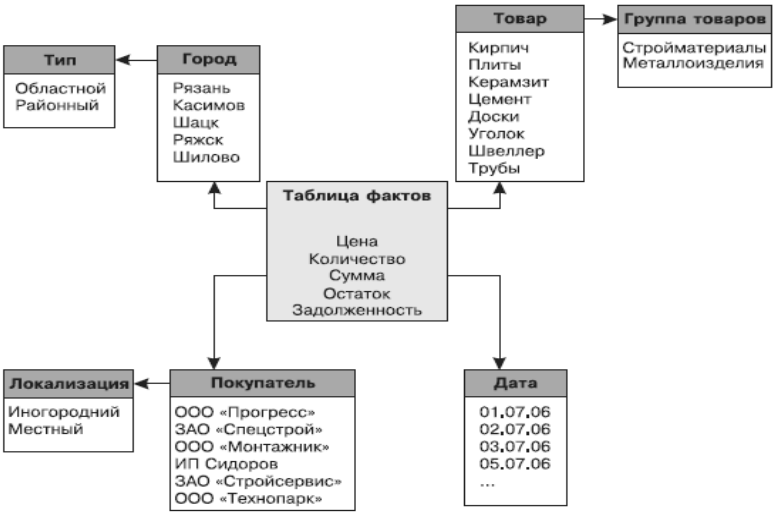

Рисунок 55. Схема «снежинка»

В сложных задачах с иерархическими измерениями может быть целесообразным использование расширенной схемы «снежинка» (Snowflake Schema). В этом случае отдельные таблицы фактов создаются для возможных сочетаний уровней обобщения различных измерений, что позволяет повысить производительность, но приводит к избыточности данных и значительным усложнениям в структуре базы данных, в которой оказывается огромное количество таблиц фактов. Также появляется необходимость поддерживать множество таблиц фактов, соответствующих каждому возможному сочетанию выбранных для запроса измерений.

Такие меры способствуют неэкономному использованию внешней памяти и увеличению времени загрузки данных в базу.

Увеличение количества таблиц фактов в базе данных обусловлено не только множественностью уровней различных измерений, но и тем, что, в общем случае, факты имеют различные множества измерений. При абстрагировании от отдельных измерений пользователь должен получать проекцию максимально полного гиперкуба, причём значения показателей в этой проекции далеко не всегда должны быть получены путём

элементарного суммирования. Поэтому при большом количестве независимых измерений необходимо поддерживать разнообразные таблицы фактов, соответствующие каждому возможному сочетанию измерений в запросе, что влечёт за собой неэкономичное использование внешней памяти, увеличивает время загрузки данных в БД схемы «звезды» из внешних источников и усложняет администрирование.

Реляционные базы данных имеют некоторые преимущества при использовании их в OLAP-системах. Во-первых, инструменты ROLAP дают возможность производить анализ над корпоративными хранилищами данных, которые в большинстве случаев реализуются с помощью реляционных СУБД. К тому же размер такого хранилища не является критическим параметром, как в случае MOLAP. Во-вторых, ROLAP-системы с динамическим представлением размерности – оптимальный выбор для задач с переменной размерностью, поскольку внесение изменений в структуру измерений не требует физической реорганизации БД. И, наконец, реляционные СУБД обеспечивают более высокий уровень защиты данных и обширные возможности разграничения прав доступа.

Главный недостаток ROLAP по сравнению с многомерными СУБД – это меньшая производительность, которая может быть сопоставима с производительностью MOLAP только при тщательной проработке схемы базы данных и настройке индексов, требующих больших усилий со стороны администраторов БД. Производительность реляционных систем приближена к производительности систем на основе многомерных баз данных только при использовании схем типа «звезда».

HOLAP. Серверы HOLAP (Hybrid Online Analytical Processing) используют гибридную архитектуру, которая объединяет технологии ROLAP (Relational OLAP) и MOLAP (Multidimensional OLAP). MOLAP работает лучше, когда данные более-менее плотные, а ROLAP показывает лучшие параметры в случае разреженных данных.

Из данной главы можно сделать несколько выводов: наиболее удобный способ представления информации для её анализа -это многомерная модель или гиперкуб, рёбрами которого являются измерения. Это позволяет анализировать данные сразу по нескольким измерениям, то есть выполнять многомерный анализ.

Измерения представляют собой последовательность значений одного из параметров, по которым проводят анализ информации. Измерения можно структурировать в виде иерархической системы. На пересечении измерений находятся количественные характеристики анализируемых фактов -меры.

Над многомерной моделью (гиперкубом) проводят операции среза, вращения, консолидации и детализации. Эти операции вместе с многомерной моделью реализуются в OLAP-системах.

OLAP (*On-Line Analytical Processing*) -технология, которая используется для оперативной аналитической обработки данных. С помощью этой технологии собирают, хранят и анализируют многомерные данные, чтобы помочь принять взвешенное решение.

Кодд разработал 12 правил для определения OLAP-систем, позднее он добавил ещё 6 правил. Все 18 правил можно разделить на четыре категории: основные особенности, специальные особенности, особенности представления отчётов и управление измерениями. В 1995 г. Пендсон и Крит разработали тест FASMI, где определение OLAP звучит как «Быстрый Анализ Разделяемой Многомерной Информации».

OLAP-системы состоят из OLAP-сервера и OLAP-клиента. OLAP-серверы можно реализовать на основе многомерных баз данных (MOLAP), реляционных баз данных (ROLAP) или скомбинировать обе модели (HOLAP).

Главные преимущества MOLAP заключаются в высокой производительности и простоте использования встроенных функций. Преимущества ROLAP в том, что эта модель позволяет работать с существующими реляционными базами данных, более экономично использовать ресурсы и иметь достаточно гибкости при добавлении новых измерений.

4.2.3 Data Warehouses

Хранилище данных (Data Warehouse) представляет собой процесс сбора данных из различных источников и управления ими с целью получения значимой бизнес-информации. Этот процесс используется для соединения и анализа бизнес-данных из разнородных источников и является ядром BI-системы, предназначенной для анализа данных и составления отчётов. База данных поддержки принятия решений (хранилище данных) существует отдельно от операционной базы данных организации. Важно понимать, что хранилище данных - это не продукт, а среда, являющаяся частью информационной системы. В хранилище данных хранится текущая и историческая информация, которая необходима пользователям для принятия решений и доступ к которой был бы затруднителен или вовсе невозможен при использовании традиционного хранилища операционных данных.

Существует несколько технологий, которые применяются для создания хранилищ данных. Одна из них -виртуальное хранилище данных. Эта технология позволяет объединить различные источники данных в единую виртуальную среду, что упрощает доступ к информации и её анализ. Другая технология -концепция Common Information Federator (CIF). Она предполагает создание единой системы, которая объединяет данные из разных источников и предоставляет пользователям доступ к этой информации по единым правилам. Это позволяет обеспечить унифицированный подход к обработке данных и упростить работу с ними.

Ещё одна технология -распределённое хранилище данных (или концепция WH Bus). Эта технология предполагает разделение данных между разными узлами сети. Каждый узел отвечает за хранение определённого вида данных или части общей информации. Такой подход позволяет повысить надёжность и доступность данных, а также облегчить их обработку.

Также существует гибридная концепция объединения технологий CIF и распределённого хранилища. В рамках этой концепции общие данные хранятся централизованно, а специализированные - распределённо на узлах сети. Это обеспечивает высокую эффективность обработки данных и при этом обеспечивает их полноту и целостность.

Виртуальное хранилище данных (облачное хранилище) (рисунок 56).

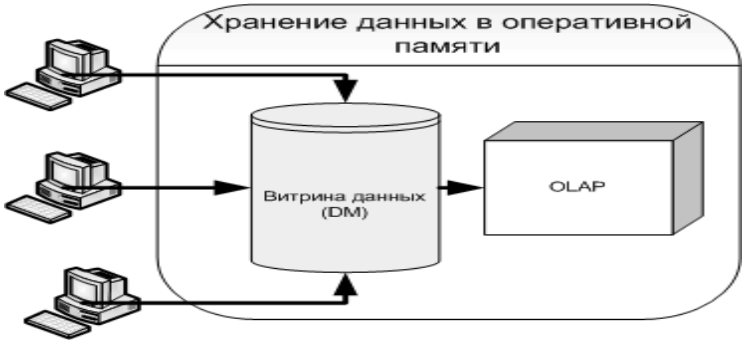

Рисунок 56. Виртуальное хранилище данных

Ключевые преимущества: быстрота работы с актуальными данными и минимизация объёма памяти на носителе. Однако есть и недостатки, среди них -необходимость постоянного доступа ко всем источникам данных, различия в представлении данных у разных источников (например, в

кодировке, числовом формате и т. д.), а также ограниченный период хранения данных у многих источников, что также ограничивает объём данных в самой DM.

Необходимость обеспечить постоянную доступность к хранилищу и увеличить период времени хранения данных привели к возникновению новых концепций их хранения.

Общее хранилище (рисунок 57).

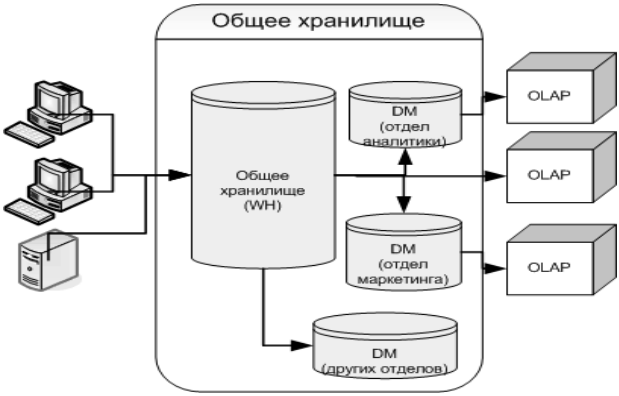

Рисунок 57. Общее хранилище

Централизованное реляционное хранилище данных в нормальной форме предоставляет следующие преимущества: атомарность данных, единые правила загрузки данных из источников, возможность сохранять большие объёмы данных, непротиворечивость данных и высокая скорость работы с данными.

К недостаткам такого хранилища можно отнести то, что организация такого хранилища данных может быть сложной задачей. Кроме того, объём хранилища данных имеет ограничения, а также существует риск дублирования данных.

Распределенное хранилище (рисунок 58).

Преимущества:

- распределённое реляционное хранилище данных позволяет хранить атомарные данные и обеспечивает непротиворечивость информации.
- реализована единая загрузка данных из внешних источников.
- есть возможность сохранять очень большие объёмы данных.

Недостатки:

- высокая сложность организации подобного хранилища данных.

- более низкая скорость работы по сравнению с общим хранилищем.

Рисунок 58. Распределенное хранилище

Гибридное хранилище.

Гибридное хранилище данных представляет собой трёхуровневую систему, в которой (рисунок 59):

1. Первый уровень -мощный SQL-сервер, где хранятся все атомарные данные.
2. Второй уровень -организация витрин данных.
3. Третий уровень -обработка данных клиентом.
4.

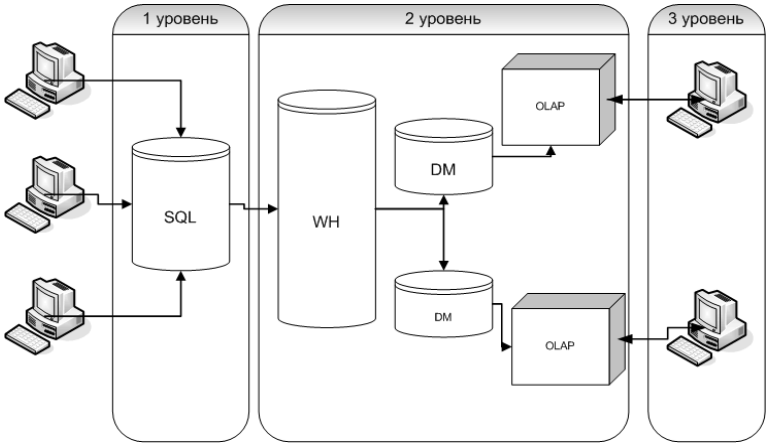

Рисунок 59. Гибридное хранилище

К преимуществам можно отнести единую загрузку данных из различных источников, возможность сохранять очень большие объёмы данных и непротиворечивость данных.

К недостаткам же относятся: высокая сложность организации хранилища и более низкая скорость работы по сравнению с общим хранилищем.

4.2.4 Не реляционные (NoSQL) базы данных

Термин «NoSQL» был впервые применён в 1998 году *Карло Строцци* как название для своей системы управления не реляционными базами данных - Strozzi NoSQL. Эта система не использовала язык SQL для манипуляций с данными, а применяла скрипты. В 2009 году *Эрик Эванс* снова использовал этот термин на конференции, посвящённой свободным распределённым базам данных, организованной *Йоханом Оскарсоном*. Встреча этой конференции также была названа «NoSQL». Именно эта конференция дала название всей технологии и положила начало буму решений NoSQL.

С этого момента термин «NoSQL» стал использоваться уже для обозначения большого количества распределённых систем управления данными, которые отказывались от поддержки ACID-транзакций - одного из ключевых принципов работы реляционных баз данных. По мнению Строцци, правильнее было бы называть эту технологию «NoREL». Термин «Not Only SQL», то есть «не только SQL» является попыткой объяснить происхождение термина, хотя такое название не совсем подходит для всех не реляционных систем.

Растущая потребность в горизонтальной масштабируемости приложений привела к появлению NoSQL-систем. Большинство из них с самого начала проектировались и создавались для работы в распределённой среде - кластере или облаке. Основные трудности связаны с использованием традиционных SQL-ориентированных систем в подобной среде.

Основной причиной отказа от поддержки транзакционной семантики явилась сложность эффективной реализации транзакций в распределенной среде: в общем случае приходилось использовать двухфазный протокол фиксации транзакций, который требовал пересылки большого количества сообщений по сети. Хотя системы NoSQL обычно не поддерживают ACID-взаимосвязи в полном объёме, в ряде случаев поддерживаются, например, атомарные операции для чтения и модификации, оптимистические блокировки и другие инструменты, помогающие упростить программирование сервисов при параллельном доступе к данным.

Не реляционная база данных представляет собой базу данных, в которой не используется табличная схема строк и столбцов, характерная для традиционных баз данных. В таких базах данных применяется модель хранения, оптимизированная под конкретный вид данных - это могут быть простые пары «ключ - значение», документы JSON или граф с вершинами и рёбрами.

Все эти форматы не используют реляционную модель. Кроме того, они обычно поддерживают определённые типы данных. Процесс запроса данных также свой для каждого вида хранилища. Например, хранилища данных временных рядов рассчитаны на запросы к упорядоченным во времени последовательностям данных, а хранилища данных графов - на анализ связей между сущностями. Ни один из форматов в полной мере не подходит для задач управления данными о транзакциях.

Термин NoSQL описывает хранилища данных, не основанные на языке запросов SQL. Вместо этого используются другие языки программирования и конструкции. На деле этот термин часто приравнивают к понятию «не реляционная база данных», хотя многие из них поддерживают совместимые с SQL запросы. Однако базовая стратегия выполнения SQL-запросов в таких системах сильно отличается от применяемой реляционными СУБД.

Хранилище пар «ключ - значение». Хранилище, в котором хранятся пары «ключ - значение», можно представить в виде большой хэш-таблицы. Каждому значению соответствует уникальный ключ, с помощью которого данные сохраняются в хранилище ключей. Этот процесс реализуется с применением функции хэширования к ключу. Выбор функции определяет равномерность распределения хэшированных ключей по хранилищу.

Большая часть хранилищ поддерживает базовые функции: запрос, вставка и удаление элементов. Для того, чтобы изменить значение полностью или частично, приложение перезаписывает существующее значение целиком. В большинстве реализаций атомарной операцией считается действие, направленное на одно значение: чтение или запись. Запись больших значений занимает продолжительное время.

Программное обеспечение хранилища не обладает информацией о значениях, которые в нём хранятся. Уровень приложения отвечает за поддержку и применение схемы данных. Приложение может сохранять в наборе произвольные данные, но стоит учесть, что некоторые хранилища ограничивают размер значений.

Фактически данные представляют собой большие двоичные объекты, которые извлекаются и сохраняются хранилищем по соответствующему ключу.

Хранилища пар «ключ - значение» адаптированы для приложений, которые выполняют простые операции поиска на основе значения ключа или диапазона ключей. Однако они не совсем подходят для систем, которым требуется запрашивать данные из нескольких таблиц хранилищ из-за сложности выполнения подобных операций.

Кроме того, функциональность хранилищ пар «ключ - значение» ограничена в сценариях, где необходима возможность запросов или фильтрации не только по ключам, но и по значению. В реляционной базе данных можно найти определённую запись с помощью предложения WHERE и отфильтровать её по не ключевым столбцам. Тогда как хранилища пар «ключ-значение» обычно не обеспечивают такой возможности поиска значений, и даже если поиск возможен, он может быть медленным.

С другой стороны, одно хранилище пар «ключ- значение» отличается простотой масштабирования. Оно позволяет эффективно распределять данные между несколькими узлами, расположенными на разных компьютерах.

Документные системы управления базами данных. Документные системы управления базами данных предлагают более широкие возможности по сравнению с системами типа «ключ-значение». Единицей хранения данных в таких СУБД является документ, представляющий собой объект с произвольным набором атрибутов (полей), например, в формате JSON. Документные СУБД обладают рядом функций - поиск по полям документов, наличие индексов, поддержка вложенных документов и массивов. Преимуществом является отсутствие заранее предопределённой схемы данных.

В отличие от систем «ключ-значение» документные СУБД позволяют выполнять запросы к коллекциям документов на основе нескольких ограничений на атрибуты. Они также могут осуществлять агрегатные запросы, сортировку результатов и поддерживают индексы на полях документов. Обычно они не гарантируют строгой согласованности ACID, но имеют свои особенности в реализации согласованности данных, поддержке атомарных операций и контроля параллельного доступа. Например, в одних системах процессы записи выполняются последовательно и только после завершения предыдущего процесса начинается следующий, обеспечивая таким образом блокировку изменений. Другие системы реализуют блокировки, которые предполагают, что изменения будут успешно применены, пока нет конфликтов между параллельными операциями.

Документные СУБД моделируют данные с гибкостью, которая в отдельных случаях оказывается более полезной и лучше сочетается с объектно-ориентированным программированием, сокращая прослойку между языком программирования и базой. Эти системы легко масштабируются. Способы построения запросов различаются, но в целом поддерживаются достаточно сложные выборки, включающие ограничения на значения полей, агрегацию, сортировку и т.д.

Чтобы обеспечить надёжность, документные СУБД используют журналирование и репликацию данных при записи на жёсткие диски или SSD-накопители. Индексы и часто используемые документы обычно хранятся в оперативной памяти для быстрого доступа. Несмотря на отсутствие транзакционной семантики на уровне нескольких документов в подобных системах, документные базы данных приближаются по функциональности к традиционным SQL-ориентированным системам управления базами. Примерами документных СУБД являются CouchDB, Couchbase, MarkLogic и MongoDB.

Базы данных на основе графов. Графовую модель данных можно считать обобщением RDF-модели или сетевой модели данных, и она представлена двумя основными элементами: узлами и связями. В зависимости от типа реализации рёбер и вершин выделяют несколько подтипов графических моделей данных.

Этот тип базы данных применяют для моделирования социальных графов, в биоинформатике и для построения семантического веба. Среди примеров таких баз данных можно назвать Neo4j, OrientDB, AllegroGraph, InfiniteGraph и FlockDB.

Некоторые авторы считают, что графовые системы управления базами данных эффективнее реляционных при работе с данными, имеющими естественную графовую структуру. Графовые СУБД также имеют преимущества в удобстве визуализации и гибкости при изменении структуры базы данных. Рассмотрим более подробно порядок работы с графовыми базами данных на примере БД **Neo4j**.

Neo4j - это NoSQL база данных, разработанная компанией Neo Technology в 2003 году. БД Neo4j определяют её как «встраиваемую базу данных, полнотранзакционный Java-движок, который хранит данные в упорядоченных графах, а не в таблицах». В Neo4j упор делается не на сами данные, которые она хранит, а скорее на связи между этими данными. Благодаря этому разнородные данные можно хранить очень просто и естественно, операции над ними требуют меньше времени. Neo4j способна

содержать десятки миллиардов узлов и столько же связей. Если есть подходящее оборудование, то она справится с задачей любого размера.

Для работы любой базой данных необходим язык запросов. Для Neo4j был специально разработан свой язык - Cypher. Это относительно простой, но мощный язык запросов, основанный на декларативном подходе.

Cypher включает в себя такие ключевые слова, как WHERE и ORDER BY, известные из SQL. Часть синтаксиса взята из таких языков программирования, как Python, Haskell и SPARQL. Соединив всё это с собственными уникальными особенностями, получили язык, позволяющий выполнять запросы к графам в простой и понятной форме.

Графовую базу данных Neo4j в её простейшей форме можно рассматривать как взаимосвязанные объекты, объединённые одним или несколькими типами отношений. Neo4j включает четыре основных элемента: узлы (или вершины) - nodes, отношения - relationships, параметры - properties и метки - labels.

Обычно каждый узел - это набор, состоящий из пар «ключ-значение». В примере, приведённом на рисунке 60, два узла представляют собой объекты: Person с ключом Name со значением «Greg» и Business с ключом Name и значением «GraphStory». Графовую базу данных Neo4j в её простейшем виде можно представить, как совокупность взаимосвязанных объектов, объединённых одним или несколькими типами отношений.

Рисунок 60. Пример графа

Свойства или параметры - это пары «ключ - значение», которые используются для представления атрибутов или метаданных о каждом узле и связи в графе. Однотипные узлы могут иметь разные свойства, и это существенное преимущество при работе с неструктурированными данными.

4.3 Универсальные форматы хранения структурированной информации

Способы хранения данных трансформировались с эволюцией информационных технологий в различных областях деятельности человека.

Создавались новые форматы данных, которые играли большую роль в оптимизации информационных процессов и обеспечения эффективной работы систем. В настоящей главе рассмотрены наиболее распространённые и универсальные форматы структурированной информации, их характеристики, преимущества и области применения.

4.3.1 Формат CSV

Формат CSV (*Comma-Separated Values*) - это текстовый формат, предназначенный для представления табличных данных, где каждая строка таблицы соответствует строке текста, содержащей поля, разделённые запятыми. Файлы в формате CSV используются для передачи большого объёма текстовых данных между различными программами и сервисами.

Структура CSV-файла. CSV-файл представляет собой простой текстовый файл, обладающий чёткой и понятной структурой не только для компьютеров, но и для людей.

Структура CSV-файла состоит из трёх основных частей (таблица 24).

Таблица 24 – Структура CSV-файла

Компонент CSV-файла	Цель
Строка заголовка	Первая строка, служащая заголовками столбцов, указывает тип данных, содержащихся в каждом столбце. В нем используется тот же формат, что и в последующих строках данных, со значениями, разделенными разделителем.
Разделитель	Символ, разделяющий значения в каждой строке. Хотя запятая является стандартным разделителем в CSV-файлах (отсюда и название), также могут использоваться другие разделители, такие как табуляция (в TSV-файлах), в зависимости от формата файла и требований.
Строки данных	После заголовка каждая строка представляет собой новую запись данных со значениями, организованными в соответствии с заголовками заголовка и разделенными выбранным разделителем. В программном обеспечении для работы с электронными таблицами эти строки соответствуют отдельным строкам в таблице, представляя данные в организованной табличной форме.

Файлы в формате CSV, за счёт своей структуры, обеспечивают простой и эффективный способ хранения и обмена табличными данными, что упрощает процессы импорта и экспорта между различными программами и платформами.

Рассмотрим пример CSV-файла, который можно увидеть в текстовом редакторе, первая строка -это строка заголовка. Она содержит тип информации, которая будет размещена в следующих строках, а также порядок, в котором эта информация представлена.

В этом примере каждая информация разделена с помощью запятой: адрес электронной почты, имя, почтовый индекс.

Это выглядит следующим образом:

1111@email.com, Директива Prime, 10111

2222@email.com, Dos Equis, 20222

3333@email.com, Tres Chic, 90210

Для упрощения работы с определённым типом файлов необходим «единый стандарт», который поддерживается в информатике и используется для более удобной работы с файлами в формате CSV. Этот стандарт называется RFC-4180 и служит для облегчения обмена данными в файлах CSV.

Работа в соответствии со стандартом RFC-4180 допускает некоторую свободу действий относительно данных, полученных от сторонних источников при обработке расширения.

Для того чтобы открыть готовый файл в формате CSV, необходимо помнить, что прочитать этот формат можно с помощью различных приложений. В первую очередь для работы с CSV могут подойти текстовые редакторы.

Открыть файл в формате CSV можно помощью различных компьютерных программ и инструментов. Чтобы открыть файл формата CSV на компьютере с операционной системой Windows, можно воспользоваться приложением Excel, которое входит в пакет Microsoft Office. Последнюю версию приложения можно загрузить через официальный сайт или приобрести лицензию. Для этого необходимо:

- запустить приложение.
- в главном меню, расположенном в верхней части экрана, выбрать раздел «Файл».
- перейти в «Открыть» («Open»). Также можно сразу перейти к загрузке документа, нажав сочетание клавиш Ctrl + O.
- выбрать «Текстовые файлы» или «Все файлы», иначе CSV-файлы не отобразятся.
- найти документ, который требуется открыть, выделить его левой кнопкой мыши и нажать кнопку Open.

Также можно сделать следующее:

- подождать, пока система загружает Excel. Желательно использовать чистый документ, чтобы исключить ошибку при открытии файла.
- перейти к разделу «Данные».
- выбрать пункт «Получение внешних данных» -«Из текста».
- найти CSV-файлы на устройстве в предложенном окне.
- кликнуть на «Import».

В окне «Мастер текстов» нужно задать формат данных «с разделителем» и включить UTF-8. Затем нужно обозначить разделитель для таблицы-CSV. Это важный этап для чтения данных. В нижней части окна «Мастера текстов» можно увидеть предварительный вид исходного текста. Далее необходимо выставить параметры для столбцов и определить область итоговой таблицы. Скорость выгрузки информации зависит от размера исходного документа. Однако CSV иногда также можно прочитать другими приложениями, в частности, text editors.

4.3.2 Формат XML

Формат XML (*eXtensible Markup Language*) представляет собой универсальный язык разметки, который был разработан для обмена структурированными данными между различными системами. Формат XML был представлен в 1998 году, и в последствии он установил новые стандарты хранения и передачи информации. XML используется для структурирования информации в виде иерархии, которая описывается с помощью тегов. Теги предназначены для определения элементов данных и устанавливают связи между ними. Каждый тег имеет своё название. Он может включать в себя атрибуты и другие элементы. Пример:

```
<название_элемента атрибут="значение">
    Содержимое элемента
</название_элемента>
```

В данном примере **<название_элемента>** и **</название_элемента>** представляют собой пару тегов, где первый является открывающим и обозначает начало элемента, а второй - закрывающим и указывает на конец этого элемента.

Основными компонентами XML-документа являются элементы, атрибуты и текстовое содержимое. **Элементы** представляют собой основные блоки данных в XML и ограничиваются начальным и конечным тегами, например,

```
<книга>
    <название>XML документ</название>
```

```
        <автор>Дмитрий Тихонов</автор>
    </книга>
```

В данном примере элемент «книга» содержит два вложенных элемента: «название» и «автор», благодаря чему предоставляется информация о книге.

Атрибуты. Это элементы XML, которые предоставляют дополнительную информацию об элементах. Они указываются внутри открывающего тега и обычно имеют вид **имя="значение"**, где имя - это название атрибута, а значение - некоторая информация, которая может быть разной в зависимости от назначения атрибута. Пример:

```
    <книга категория="Программирование и обработка данных">
        <название>Обработка данных</название>
        <автор>Дмитрий Тихонов</автор>
    </книга>
```

Элементы также могут содержать текст:

```
    <книга>
        <название> Обработка данных </название>
        <автор> Дмитрий Тихонов </автор>
        <описание>Эта книга предназначена для изучающих обработку данных </описание>
    </книга>
```

Одним из ключевых преимуществ XML является возможность поддерживать схемы, что позволяет определять строгую структуру данных и проводить их валидацию. XML-схема представляет собой набор правил и ограничений, которым должен соответствовать XML-документ, чтобы он был признан валидным. Она определяет допустимые элементы, атрибуты, порядок и уровень вложенности элементов, а также типы данных для текстового содержимого.

Для описания схем в XML используются два основных подхода: DTD (Document Type Definition) и XSD (XML Schema Definition). DTD, или определение типа документа, представляет собой структуру, которая может быть указана в самом XML-документе или размещена отдельно. DTD позволяет указать разрешённые элементы, их взаимоотношения, атрибуты и их типы данных. Рассмотрим пример использования DTD для определения структуры простейшего XML-документа о книжном магазине:

```
    <!DOCTYPE bookstore [
        <!ELEMENT bookstore (book+)>
        <!ELEMENT book (title, author, price, description?)>
        <!ELEMENT title (#PCDATA)>
```

```
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT price (#PCDATA)>
    <!ELEMENT description (#PCDATA)>
]>
```

В этом примере мы определили элемент <bookstore>, который может включать один или несколько элементов <book>. Каждая книга, обозначаемая элементом <book>, состоит из следующих элементов: <title>, <author>, <price>. Также книга может содержать дополнительный элемент <description>.

XSD (XML Schema Definition) является более продвинутым и универсальным подходом к организации структуры данных в формате XML по сравнению с DTD. Его основное преимущество - способность выполнять более детальную и обширную проверку данных в XML-документе.

XSD представляет собой отдельный документ, написанный на языке XML. Этот стандарт поддерживает большое количество типов данных, что позволяет более точно представлять структуру документа. С помощью XSD можно легко устанавливать ограничения на типы данных, атрибуты и степень вложенности элементов. Пример:

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="книга">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="название"
type="xs:string"/>
        <xs:element name="автор" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

В этом примере при помощи XSD определён элемент «книга», который включает в себя два элемента. XSD также точно указывает, что оба эти элемента имеют тип данных «строка».

XML - это расширяемый язык разметки, который широко применяется в различных областях благодаря своему гибкому формату. Одним из основных применений XML является его использование в качестве стандартного формата обмена информацией между веб-сервисами. Это связано с тем, что XML поддерживает сложные структуры и может быть использован для представления любых данных, которые можно описать в виде древовидной структуры. Пример использования XML в этой области -

SOAP (Simple Object Access Protocol), который представляет собой стандартный протокол обмена сообщениями на основе XML. Веб-сервисы используют SOAP для отправки информации через сеть в формате XML. Кроме того, Amazon Web Services (AWS) активно используют XML в своих API: большинство сервисов AWS предоставляют API на основе этого формата. Эта практика включает в себя S3 (для хранения и извлечения объектов), EC2 (для управления виртуальными серверами), SimpleDB (для работы с базами данных) и несколько других сервисов.

В контексте взаимодействия с другими ресурсами язык XML применяется для синдикации контента (распространение информации с одного сайта на большее количество ресурсов). Если говорить о XML, то в данном случае это его использование в форматах RSS (*Really Simple Syndication*) и Atom для новостей и блогов. Такая практика предоставляют пользователям возможность подписаться на обновления сайтов. При этом сами сайты «упаковывают» свои обновления в универсальный формат XML, доступный для чтения различными клиентскими приложениями - так называемым агрегаторам новостей.

Само составление макетов документов и файлов стало ещё одной областью применения XML. Так, программы Microsoft Office (например, Word и Excel) используют формат Office Open XML для своих файлов. Также стоит отметить Scalable Vector Graphics (SVG), представляющий собой XML-формат для описания двумерной векторной графики.

Во многих случаях конфигурационные данные удобно хранить в XML благодаря его структурированному и легкочитаемому формату. Хорошим примером является использование XML-файла для указания свойств соединения с базой данных, маппинга классов и других настроек в Hibernate - фреймворке для объектно-реляционного отображения на Java.

В Spring Framework конфигурационный файл XML служит для определения beans (объектов), которые управляются Spring IoC контейнером.

4.3.3 Формат HTML

Язык гипертекстовой разметки HTML (*Hyper Text Markup Language*) - это язык, который используется для создания веб-документов. Первые упоминания о гипертексте встречаются ещё во времена создания первых рукописных книг. На полях таких книг иногда делали сноски, например, «см. такую-то страницу», что и было первыми неудобными гипертекстовыми ссылками.

Понятие «гипертекст» впервые употребил учёный *Т. Нильсон* в 60-е годы XX века, описав свой проект «Ксанаду», цель которого заключалась в создании электронных библиотек с использованием гипертекстовых ссылок. Суть проекта заключалась в том, что библиотеки должны были храниться на спутнике и управляться с помощью радио и телефонной связи для выполнения перехода с одного документа на другой.

На основе языка разметки документов SGML в начале 90-х годов *Тим Бернерс-Ли* создал HTML. Однако есть один нюанс - если бы не *Марк Андриссен* и группа студентов из Иллинойского университета, которые примерно в это же время разработали первый браузер Mosaic и запустили его бесплатное распространение, то HTML так и остался бы теорией.

Браузер Mosaic демонстрировал гипертекстовые страницы, и с этого момента началось развитие веб-страниц и веб-серверов.

Благодаря своим достоинствам язык HTML стал повсеместно использоваться, поэтому возникла необходимость создания стандартов для его применения, чтобы предотвратить разночтение в будущем. С этой целью Консорциум W3C разработал и утвердил первую спецификацию HyperText Markup Language. С течением времени в язык вносились изменения и появлялись новые спецификации. В настоящее время во всём мире используется HTML5.

HTML - это текстовый файл, который содержит набор команд гипертекстовой разметки. Эти команды указывают браузеру, как правильно отображать содержимое документа. В таком документе не всегда содержится текст, но структура должна присутствовать обязательно. Для использования документа в интернете он должен иметь расширение **.htm** или **.html**. Работать с такими файлами можно в обычном блокноте Windows, однако удобнее использовать веб-редакторы. Они позволяют создавать страницы с форматированием, изображениями и другими элементами.

Базовые элементы и команды языка HTML называются ***тегами*** (tag), они регистронезависимы и всегда находятся в угловых скобках. Тег имеет следующую структуру: `<тег параметр1 = «значение» параметр2 = «значение»`, где параметры являются атрибутами тега.

Параметры (или атрибуты) указывают тегу на выполнение определённых функций или содержат дополнительную информацию. Например, тег `<img>` используется для вставки графических изображений на веб-страницу и может иметь параметры width и height, указывающие ширину и высоту изображения. Пример:

```
<img src="berlin.jpg" width="500" height="375">
```

Теги в языке HTML делятся на две основные группы: одиночные и контейнерные (парные). Одиночные теги, такие как <p> или <b>, используются отдельно и не требуют закрытия. Контейнерные же теги состоят из открывающего и закрывающего элементов и могут вкладываться друг в друга как матрёшки.

Любая веб-страница, отображаемая в браузере, имеет чёткую структуру, которая представляет собой сочетание HTML-элементов и текста. **Заголовок и тело страницы** располагаются между тегами **<html>** и **</html>**, которые служат для обозначения начала и конца HTML-документа. Это сообщает браузеру о типе документа и способе его обработки, так как существуют различные версии HTML-стандартов, а также другие технологии, например, XHTML и XML. В HTML-документе должны быть элементы HEAD, где содержится информация о самом документе, и BODY, который является видимой частью веб-страницы и содержит всё её наполнение.

Первым тегом в документе согласно стандартам, W3C должен стоять тег <html>, который указывает на тип документа - DTD (определение типа документа) и даёт команду браузеру, как обрабатывать текущую страницу. Раздел документа head определяет его заголовок и содержит дополнительную информацию о документе для браузера. Этот раздел не является обязательным, но рекомендуется использовать его в HTML-документах, так как правильно составленный заголовок может быть полезен.

Раздел заголовка начинается тегом <head> сразу после тега <html> и включает другие элементы заголовка между открывающим и закрывающим тегами элемента head. **Название HTML-документа** задаётся с помощью тега <title>, оно отображается в заголовке окна браузера. Название документа записывается как строка текста между тегами <title> и </title>. Длина этого текста может быть произвольной, но рекомендуется ограничивать её 60 символами. Тег <title> должен находиться исключительно в разделе head.

Раздел тела документа. В этом разделе содержится информация, которая отображается в окне браузера. Раздел body должен начинаться тегом <body> и заканчиваться тегом </body>. Между этими тегами располагаются элементы HTML, которые составляют содержание документа. Тег <body> определяет внешний вид документа и имеет ряд атрибутов:

- атрибут **text** задает цвет текста всего документа в формате RGB или в символьной нотации. По умолчанию используются настройки браузера;

• атрибут **bgcolor** задает цвет фона окна браузера документа также в формате RGB или символьной нотацией, по умолчанию - используются настройки браузера;

• атрибут **background** позволяет указать адрес фонового рисунка и имя к нему. Этот рисунок будет размножен и распределен на заднем плане документа;

• атрибуты **link**, **vlink** и **alink** задают цвета гиперссылок в формате RGB или в символьной нотации

Пример документа HTML-документа:

```
<html>
<head>
 <title>Приветствие</title>
</head>
<body>
 <P>Добро пожаловать! </P>
</body>
   </html>
```

Чтобы задать цвет различных элементов HTML-документа, необходимо указать значение соответствующих атрибутов. Значения этих атрибутов можно задать двумя способами:

• определить цвет в формате RGB - это метод, который использует три числа для представления красного, зелёного и синего компонентов цвета;

• использовать символьную нотацию - указание названий определённых цветов, таких как «red» (красный), «green» (зелёный) и т. д.

Оба подхода позволяют точно задать цвет визуальных элементов HTML-страницы.

Формат RGB - это система указания цвета, основанная на смешении трёх основных цветов: красного (RED), зелёного (GREEN) и синего (BLUE), итоговый цвет определяется цифрами в шестнадцатеричном коде. Значения красного, зелёного и синего цвета задаются шестнадцатеричными цифрами в диапазоне от 0 до FF, что в десятичной системе соответствует диапазону от 0 до 255. Затем эти значения объединяются в одно число, перед которым ставится символ #. Таким образом, можно описать более шестнадцати миллионов оттенков цвета. Например, цвет фиолетового оттенка может быть указан в формате RGB числом #800080. Благодаря использованию данного формата возможно определение и указание широкой цветовой гаммы.

У задания цвета в формате RGB есть один недостаток - необходимо помнить определённые совокупности цифр для указания того или иного

цвета. В отличие от этого, в символической нотации таких недостатков нет - можно просто указывать название цвета на английском языке. Однако у этого способа указания цвета тоже есть недостаток - всего определено шестнадцать имён цветов. Соответствие формата RGB и символьной нотации (таблица 25).

Таблица 25 - Соответствие указания цвета в символьной нотации и формате RGB

Символьная нотация	**Формат RGB**	**Цвет**
Black	#000000	Черный
Silver	#C0C0C0	Серебро
Gray	#808080	Серый
White	#FFFFFF	Белый
Maroon	#800000	Темно-бордовый
Red	#FF0000	Красный
Purple	#800080	Фиолетовый
Fuchsia	#FF00FF	Розовый
Green	#008000	Зеленый
Lime	#00FF00	Известь
Olive	#808000	Оливковый
Yellow	#FFFF00	Лимонный
Navy	#000080	Темно-синий
Blue	#0000FF	Синий
Teal	#008080	Темно-бирюзовый
Aqua	#00FFFF	Бирюзовый

HTML, или язык разметки гипертекста, является важным инструментом для создания веб-сайтов, веб-страниц и веб-приложений. Он обладает рядом преимуществ, таких как простота использования и широкая поддержка инструментами разработки. Эти факторы делают HTML привлекательным выбором для бизнеса, управления проектами и других сфер по сравнению с другими методами разработки программного обеспечения.

Этот язык особенно удобен для создания простых веб-страниц или сайтов малого и развивающегося бизнеса, так как не требует значительных инвестиций в лицензии и программное обеспечение, а также сложного программирования, зато обеспечивает простое и быстрое размещение информации в сети.

4.3.4 Формат JSON

Формат JSON (*JavaScript Object Notation*) - это облегчённый формат обмена данными, который удобен как для людей, так и для машин. Он был

основан на подмножестве стандарта языка программирования JavaScript ECMA-262 в третьем его издании (декабрь 1999 г.) и является текстовым форматом, полностью независимым от языка, но использующим соглашения, известные программистам языков семейства C, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других.

JSON построен на двух структурах:

• ***набор пар ключ/значение***, которые в различных языках представлены как объект, запись, структура, словарь, хэш-таблица, список с ключами или ассоциативный массив (рисунок 61).

Рисунок 61. Структура ключ/значение JSON

• ***упорядоченный список значений***, обычно реализуемый в языках как массив, вектор, список или последовательность.

Формат JSON поддерживает различные типы информации, которые можно посмотреть в таблице 26.

Таблица 26 - Типы данных в JSON

Тип	**Описание**
Число	С плавающей точкой двойной точности в JS.
Строка	Unicode, содержащий двойные кавычки и обратную косую черту.
Логический	True/False.
Массив	Последовательность значений упорядоченного характера.
Значение	Строка, число, истина, ложь, нуль и так далее.
Объект	Набор пар «ключ/значение» неупорядоченного характера.
Пробел	Применяется между несколькими парами токенов.
Ноль (null)	Обозначение пустоты.

Это универсальные структуры данных, которые поддерживаются практически всеми современными языками программирования в той или иной форме. Важно, чтобы формат данных, который можно использовать совместно с разными языками программирования, также основывался на этих структурах. В формате JSON эти структуры представлены следующим образом:

объект представляет собой неупорядоченный набор пар имя/значение, он начинается с *левой фигурной скоб*ки **{** и заканчивается *правой фигурной скобкой* **}**. За каждым именем следует *двоеточие* **:**, а пары ключ/значение разделяются *запятой* **,**. Схематично порядок записи объекта представлен на рисунке 62.

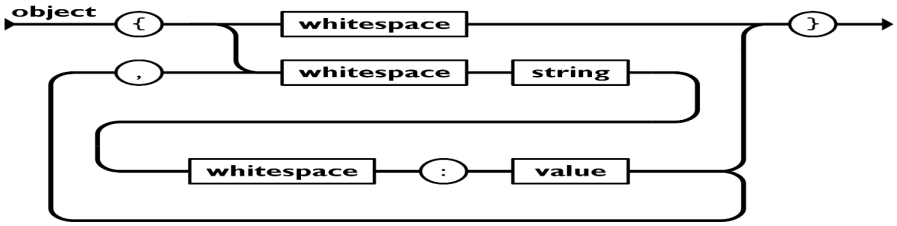

Рисунок 62. Объект

массив представляет собой упорядоченный набор значений, который начинается с *левой квадратной скобки* **[**и заканчивается *правой квадратной скобкой* **]**, а значения в нём разделяются запятыми (рисунок 63).

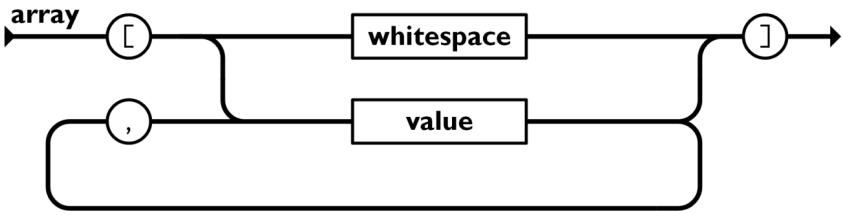

Рисунок 63. Массив

Значением может быть *строка*, заключённая в двойные кавычки, *число*, логические значения true (истина) или false (ложь), значение null, *объект* или *массив*. При этом описанные структуры могут быть вложенными (рисунок 64).

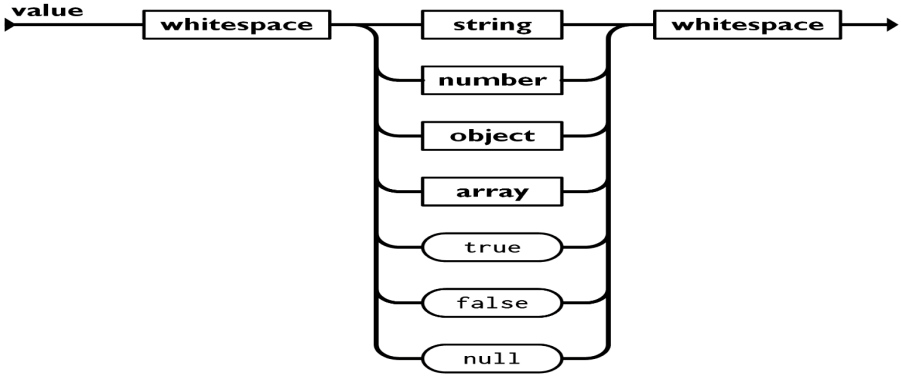

Рисунок 64. Значение

Строка - это последовательность из нуля или более символов Юникода, которая заключена в двойные кавычки с использованием экранирования обратной косой черты. Символ представлен в виде строки из одного символа. Очень похоже на то, как устроена строка в языках C или Java (рисунок 65).

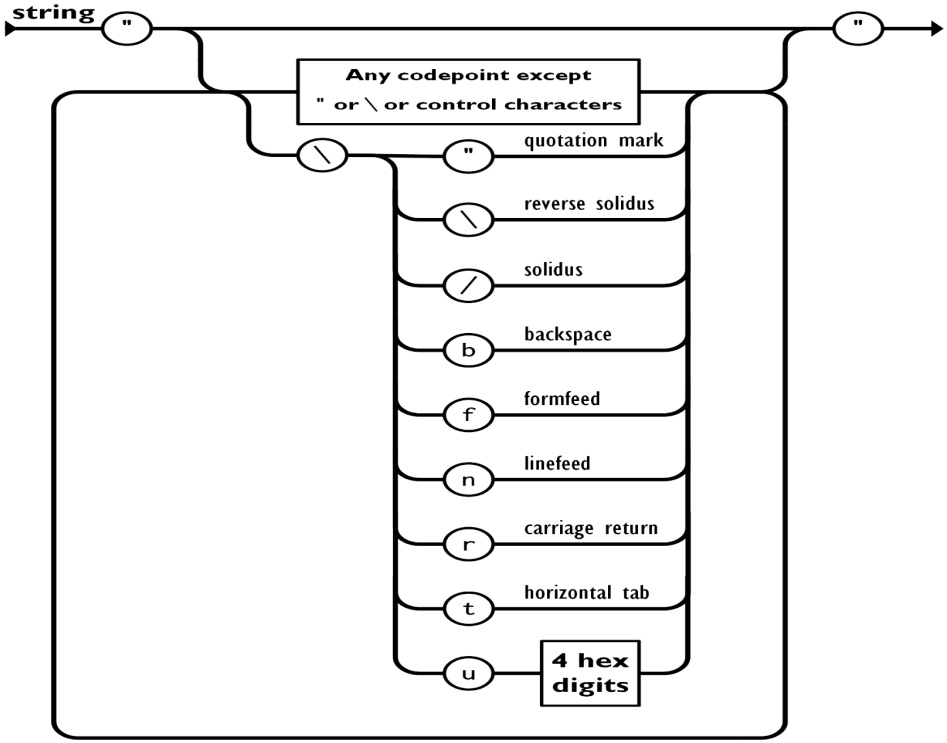

Рисунок 65. Строка

Число в общем аналогично числу в языках C и Java, но для него не используются восьмеричный и шестнадцатеричный форматы (рисунок 66).

Большинство веб-сервисов используют JSON для обмена данными через API. В RESTful API обычно данные отправляют в формате JSON. Например, с помощью YouTube Data API можно запросить информацию о видео, указав его идентификатор. В ответ получаем объект JSON со всеми данными видео. Тот же принцип применим и к GitHub API: можно запросить информацию о репозиториях, пользователях, коммитах и других аспектах платформы - данные вернут в формате JSON.

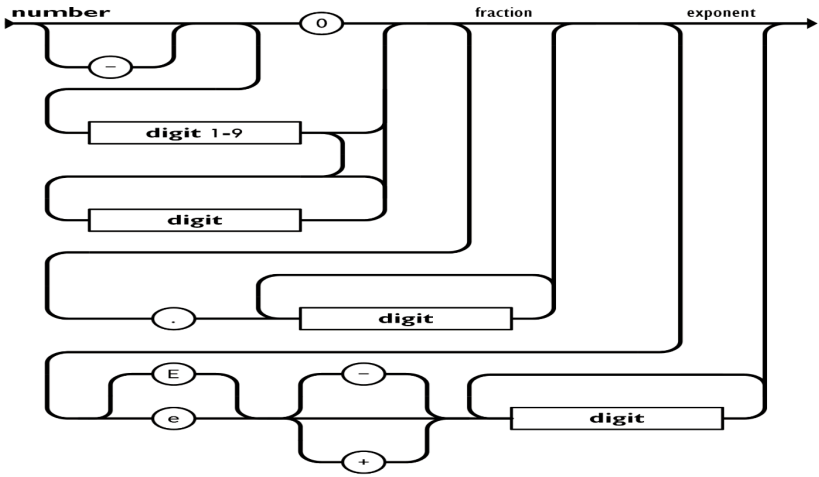

Рисунок 66. Число

Формат JSON поддерживается многими типами баз данных. MongoDB использует BSON - бинарную форму JSON - для построения документов. Некоторые классические SQL БД, такие как PostgreSQL или MySQL, также поддерживают работу с JSON: эти базы позволяют хранить JSON в виде строк и предлагают встроенные функции для работы с данными.

JSON также используют для конфигурационных файлов в веб-приложениях. Если написать на Node.js, файл package.json содержит данные о вашем проекте и его зависимостях. Не менее важен файл webpack.config.js для сборщика модулей Webpack: он также содержит JSON-подобные объекты.

Когда пользователи отправляют запросы к серверу (например, заполняют и отправляют форму), сервер обычно отвечает в формате JSON, чтобы клиент мог легко обработать эти данные и отобразить их на фронте.

Работа с такими запросами не вызывает затруднений: у большинства популярных веб-браузеров и серверных языков программирования (таких как Node.js) есть встроенные инструменты для конвертации данных из JSON в объекты JavaScript и обратно.

При создании графиков и диаграмм данные часто структурируют с помощью JSON. Визуализировать структурированные данные позволяют такие библиотеки, как D3.js, Plotly и Highcharts. Однако конкретные методы структурирования данных зависят от выбранной библиотеки.

JavaScript позволяет без труда работать с данными в формате JSON: объекты JavaScript аналогичны формату JSON, что упрощает их обработку.

JSON позволяет воссоздать структурированные тестовые данные для приложений, поэтому широко применяется при тестировании веб-продуктов.

Стоит отметить, что формат JSON не единственный, кто способен решить подобные задачи. Всё больше пользователей выбирают его в качестве альтернативы более привычному XML - и всё же оба формата остаются незаменимыми инструментами для разных сценариев.

Контрольные вопросы по главе 4

1. Что такое структурированные данные?
2. Дайте определение понятию «файл».
3. Какие могут быть форматы файлов. Охарактеризуйте каждый формат.
4. Дайте определение понятию «файловая система».
5. Перечислите основные функции файловых систем.
6. Перечислите и охарактеризуйте основные файловые системы Windows.
7. Перечислите и охарактеризуйте основные файловые системы macOS.
8. Перечислите и охарактеризуйте основные файловые системы Linux.
9. Какие операции можно выполнять над файлами?
10. Дайте определение понятию «база данных».
11. Дайте определение понятию «СУБД».
12. Перечислите и охарактеризуйте основные свойства баз данных.
13. Перечислите 13 правил реляционной модели.
14. Перечислите операций реляционной алгебры. На какие две класса они делятся?
15. Опишите структуру языка SQL.
16. Дайте определение понятию «многомерный анализ».
17. Перечислите и охарактеризуйте архитектуры OLAP.
18. Дайте определение понятию «Хранилище данных».
19. Перечислите и охарактеризуйте технологии хранилищ данных.
20. Охарактеризуйте формат CSV. Какими возможностями он обладает?
21. Охарактеризуйте формат XML. Какими возможностями он обладает?
22. Охарактеризуйте формат HTML. Какими возможностями он обладает?
23. Охарактеризуйте формат JSON. Какими возможностями он обладает?

Глава 5. ТЕХНОЛОГИИ ОБРАБОТКИ ДАННЫХ

С 1960-х годов информационно - коммуникационные технологии (ИКТ) прошли путь от простых систем обработки файлов до сложных и мощных систем управления базами данных.

Исследовательская деятельность в области СУБД началась в 1970-е годы и сместилась от ранних иерархических и сетевых баз данных к реляционным системам управления базами данных. Кроме того, были разработаны инструменты моделирования данных и вопросы индексирования и организации данных.

Пользователи получили гибкий и удобный интерфейс доступа к данным с помощью языков запросов типа SQL, пользовательские интерфейсы, управление транзакциями. Созданные и поддерживаемые базы данных имели преимущественно ограниченный регистрирующий характер и гарантировали выполнение рутинных операций линейного персонала. Основными требованиями к таким системам были обеспечение транзакционности и оперативность выполнения изменений.

В середине 80-х годов началась популяризация технологии баз данных. Исследователи сосредоточились на новых, все более мощных СУБД. Появились новые модели данных: объектно-ориентированные, объектно-реляционные и дедуктивные. Возникали предметно-ориентированные базы данных и СУБД - пространственные, временные, мультимедийные и научные. Эффективные методы OLTP внесли большой вклад в эволюцию и широкое внедрение реляционной технологии в качестве инструмента для эффективного хранения, извлечения и управления большими объемами структурированных данных реляционных СУБД.

С развитием Интернета возросла интенсивность формирования и архивирования данных, что способствовало развитию масштабируемых программно-аппаратных комплексов, дорогостоящих мощных и недорогих пользовательских компьютеров, и накопителей данных. Это способствовало взрывному росту индустрии ИКТ и сделало большие объёмы разнородных данных доступными для хранения и управления в них транзакциями.

При этом росла потребность анализировать данные в разновременном аспекте и строить произвольные запросы, обрабатывая большие объёмы данных, полученных из различных регистрирующих БД. Использование традиционных регистрирующих систем и БД становилось затруднительным, так как информация остается актуальной только на момент запроса к базе данных. Интерфейсы данных систем рассчитаны на

проведение определенных стандартизированных операций, а возможности получения результатов на произвольный запрос ограничены.

Возможности обработки больших массивов данных также могут быть ограничены вследствие ориентации СУБД на нормализованные данные. Ответом на возникшую потребность стало появление новой технологии организации баз данных - технологии хранилищ данных (Data Warehouses), предполагающих предварительную обработку данных и их интеграцию, а также онлайновую аналитическую обработку (OLAP).

Несмотря на явные преимущества такого инструмента анализа данных, он ориентирован на хорошо структурированные табличные данные и не включает использование дополнительных методов аналитики, таких как классификация, кластеризация, регрессионный анализ, моделирование, прогнозирование и интерпретация многомерных данных и т. д.

Сегодня наблюдается высокий уровень развития масштабируемой аппаратно-программной информационно-коммуникационной технологии, которая позволяет увеличивать и так значительные архивы данных. Существует значительный задел в области компьютерных наук и информационных технологий, разработаны теоретические и практические аспекты теории вероятности и математической статистики. Однако стоит отметить, что существует ощутимый избыток данных при нехватке информации и знаний.

Быстрорастущие объёмы накопленных и пополняемых автоматически данных превышают способности человека в их эффективной обработке, поэтому важные решения часто принимаются на основе интуиции, а не на основе аналитических выводов из информативных баз данных. В результате этого большие объёмы данных становятся своеобразными «необработанными могилами».

Вследствие этого в последние годы стремительно развивается область Data Mining, нацеленная на поиск и разработку методов извлечения знаний из имеющихся данных, которые позволяют принимать конкретные и обоснованные управленческие решения.

На рисунке 67 представлен пример обобщённого иерархического представления методологий обработки данных. Методологии начинаются с интеграции разнородных источников данных и заканчиваются использованием методов Data Mining для принятия управленческих решений.

Рисунок 67. Пример обобщенного иерархического представления методологий обработки данных при принятии управленческих решений

5.1 Технологии анализа больших объемов данных (Big Data)

Большие данные (Big Data) - это тема, которая вызывает определённые дискуссии в научном сообществе. Но, несмотря на отсутствие однозначного определения данного понятия, можно выделить два определения, которые описывают общую концепцию и с которыми согласятся большинством людей. **Первое определение** было предложено *Мервом Адрианом* из компании Gartner в статье для журнала Teradata Magazine в первом квартале 2011 года: «**Большие данные** - это данные, сбор, управление и обработку которых невозможно осуществить с помощью наиболее часто используемых аппаратных сред и программных инструментов в течение допустимого для пользователя времени». **Другое определение** появилось в докладе *McKinsey Global Institute* в мае 2011: «**Большие данные** - это наборы данных, размеры которых выходят за пределы возможностей по сбору, хранению, управлению и анализу, присущих обычному программному обеспечению базы данных».

Очевидно, что критерии того, какие данные считаются «большими», могут меняться с течением времени. Конкретное наполнение понятия зависит от уровня технологического развития в конкретный момент

времени, от финансовых и других возможностей организаций или даже отраслей.

Также стоит отметить, что отсутствие фиксированных критериев вызывает настороженность у некоторых специалистов, изучающих проблематику больших данных. Понятие «большие данные» подразумевает не только их объём, согласно Gartner Group, слово «большие» также относится и к некоторым другим характеристикам источника больших данных. Это не только возросший объём, но и скорость передачи и разнообразие источников, что усложняет работу с большими данными, поскольку приходится иметь дело не просто с большим количеством данных, но с тем, что они поступают к вам очень быстро, в сложных формах и из разнообразных источников.

Большие данные отличаются от традиционных данных рядом характеристик. Одна из таких характеристик заключается в том, что данные могут быть созданы автоматически машиной без участия человека. Так, встроенные в двигатель датчики могут генерировать данные в процессе работы, датчик отслеживания местоположения в смартфоне - записывать передвижение, а «умные» устройства - собирать статистику использования. В отличие от этого, традиционные данные всегда предполагают присутствие человека: покупки, звонки, оплата счетов или доставка товаров не обходятся без участия людей.

Вторая характеристика заключена в появлении принципиально новых источников данных. Они не просто добавляют информацию к существующим источникам. К примеру, технологические изменения в бизнесе, такие как совершение покупок через интернет, не приводят к созданию новой информации. Однако, наблюдение за действиями пользователей в Интернете может дать новые знания о поведении и предпочтениях потребителей. А переход со снятия показаний счётчика вручную к автоматическому сбору данных с интеллектуального счётчика может предоставить совершенно иную информацию.

Новая информация может быть результатом появления новых технологий, которые записывают большое количество данных автоматически.

Третья характеристика - некоторые источники больших данных изначально не предназначались для создания дружественных форматов предоставления информации для обеспечения лёгкого анализа. С данными из социальных сетей трудно работать из-за ошибок и проблем с понятностью. Традиционные же системы, например, системы отслеживания

транзакций, предоставляют данные в форматах, облегчающих их загрузку и анализ.

Потоки больших данных не всегда представляют особую ценность, большая часть данных может оказаться бесполезной. В журналах логов содержится как очень полезная информация, так и данные, не имеющие ценности. Необходимо отсортировать ненужные фрагменты информации и извлечь ценные и релевантные. Традиционные источники данных сразу разрабатывались так, чтобы содержать сто процентов релевантных данных. Это было связано с ограничениями масштабируемости - слишком дорого обходилось бы включение в поток данных чего-то неважного. Мало того, что записи данных были предопределены заранее, каждый фрагмент информации имел высокую ценность. С тех пор объём носителей перестал быть ограничением - потоки данных включают всю возможную информацию. Позже приходится разбираться, что же из этих данных имеет значение. Такой подход гарантирует, что ничего не будет упущено, но усложняет процесс анализа больших данных.

Дуг Лейни из Meta Group (входит в состав Gartner) ещё в 2001 году обратил внимание на необходимость решения вопросов, связанных с хранением и обработкой информации в контексте использования больших данных. В результате были определены три направления, на которых нужно сосредоточить усилия:

Volume - объём. В рамках работы с большими данными информацию постоянно получают, хранят и обрабатывают. Её объём постоянно меняется - имеющаяся информация обновляется, к ней добавляется новая. При работе с большими массивами данных важно иметь возможность оперативно увеличивать доступные ресурсы, так как входящий поток данных потенциально может увеличиваться.

Velocity - скорость. Большое значение имеет скорость обработки данных, которая соответствует целям проекта. Например, огромное количество датчиков фиксируют информацию о сейсмических изменениях на территории конкретной страны или в мире. Эти данные поступают в центр обработки данных (ЦОД), где выполняется их анализ. Если по какой-либо причине данные будут обрабатываться часами вместо того, чтобы сделать это за минуты, то в случае получения информации о землетрясении превентивные меры не получится принять вовремя. Последствия могут быть катастрофическими.

Компания IDC добавила к параметрам V ещё один - **Value**, или ценность информации. Показатель отражает актуальность и достоверность данных.

Ценность также может быть утрачена, если информация потеряет свою актуальность раньше, чем ею смогут воспользоваться. Это называется **Validity** - период полезного действия информации. Таким образом, несвоевременное получение данных может полностью обесценить их. Становится понятно, что скорость обработки поступающих данных имеет большое значение, в противном случае можно упустить их ценность и передать на дальнейший анализ устаревшие данные или предоставить неактуальную информацию.

При этом крайне важно, чтобы скорость обработки данных и хранилища могли легко масштабироваться при необходимости. Это также является одним из принципов технологии Big Data.

Как уже говорилось ранее, наряду со структурированными данными существует информация, которая поступает в неструктурированном виде, и она доминирует в общем потоке информации. Это является пятым параметром: **Variety** - разнообразие.

Таким образом описательную модель Big Data можно представить следующим образом (рисунок 68).

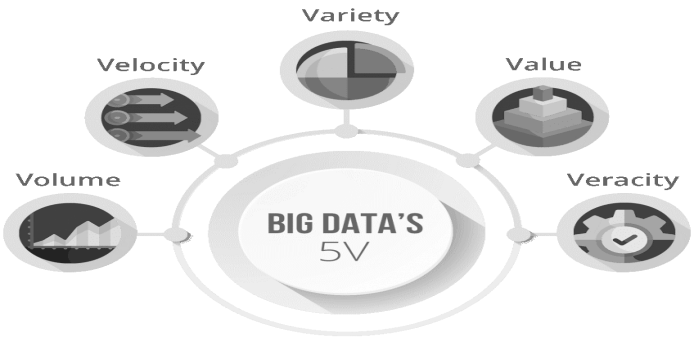

Рисунок 68. Описательная модель Big Data

Большие данные сравнивают с приливной волной, а её приручение - настоящий вызов. Методы, процессы и системы анализа, внедрённые в организациях, будут использоваться до предела, а возможно, и сверх предела. Необходимо разработать дополнительные методы и процессы анализа на базе обновлённых технологий и методов, чтобы эффективно анализировать большие данные и действовать на основании полученных результатов.

В настоящее время существует множество разнообразных методик анализа массивов данных, при этом исследователи продолжают работать над созданием новых методик и совершенствованием уже существующих. В основе этих методик лежит инструментарий, заимствованный из статистики и информатики. К основным техникам и методам анализа и обработки больших данных можно отнести следующие:

1. ***Интеллектуальный анализ информации или Data Mining***, что можно перевести как «добыча данных» или «раскопка данных», имеет несколько определений, но в общем смысле это процесс поддержки принятия решений, основанный на анализе данных с целью выявления в них скрытых закономерностей и шаблонов информации. Технология Data Mining предназначена для поиска в больших массивах данных неочевидных, объективных и практически полезных закономерностей.

Методы Data Mining используются, как правило, для выполнения четырёх основных типов задач: классификация, регрессия, кластеризация и ассоциация. Рассмотрим каждую из них.

Классификация. При анализе данных часто возникает необходимость определить, к какому из известных классов относятся исследуемые объекты, т. е. классифицировать их. Например, когда человек обращается в банк за кредитом, банковский служащий должен решить, кредитоспособен ли потенциальный клиент или нет. Это решение принимается на основании данных об исследуемом объекте: его месте работы, размере заработной платы, возрасте, составе семьи и т.п. В результате анализа этой информации банковский служащий относит человека к одному из двух классов: «кредитоспособен» или «некредитоспособен».

Другой пример задачи классификации – это фильтрация электронной почты. Программа фильтрации должна классифицировать входящее сообщение как спам (нежелательная почта) или как письмо. Решение принимается на основании частоты появления в сообщении определённых слов: имени получателя, безличного обращения, слов «приобрести», «заработать», «выгодное предложение» и др.

Количество классов в задачах классификации может быть более двух. Так, в задаче распознавания образа цифр таких классов может быть 10 (по количеству цифр в десятичной системе счисления). Объектом классификации является матрица пикселов, представляющая образ распознаваемой цифры. Цвет каждого пиксела является характеристикой анализируемого объекта.

Задачу классификации в Data Mining рассматривают как задачу определения значения одного из параметров объекта на основании значений

других параметров. Определяемый параметр часто называют зависимой переменной, а параметры, участвующие в его определении -независимыми переменными. В рассмотренных примерах независимыми переменными были следующие: место работы, размер заработной платы, возраст, состав семьи для получателя кредита, частота появления в почтовом сообщении определённых ключевых слов для задачи фильтрации электронной почты. Задача классификации является задачей определения значения одного из параметров анализируемого объекта на основании значений других параметров. Параметр, значение которого определяется, часто называют зависимой переменной, а параметры, участвующие в его определении - независимыми переменными.

В качестве независимых переменных в рассмотренных примерах были использованы зарплата, возраст, количество детей и т.д., частота определённых слов, значения цвета пикселов матрицы. Зависимыми переменными в этих же примерах являлись:

- кредитоспособность клиента (возможные значения этой переменной - «да» и «нет»);
- тип сообщения (возможные значений этой переменной -«спам» или «обычное сообщение»);
- цифра образа (возможные значение этой переменной-от 0 до 9).

Регрессия. Во всех рассмотренных случаях независимая переменная принимала значение из конечного множества значений: {да, нет}, {спам, обычное сообщение}, {0, 1, ..., 9}. Если значениями независимых и зависимой переменных являются действительные числа, то задача называется задачей регрессии. Примером задачи регрессии может служить задача определения суммы кредита, которую банк может выдать клиенту.

Задача классификации и регрессии включает в себя два этапа. **На первом этапе** создается обучающая выборка, которая содержит объекты с известными значениями как независимых, так и зависимых переменных. Такие данные могут включать информацию о клиентах, получивших кредиты на разные суммы (с известными данными об их погашении), сообщения, классифицированные вручную как спам или не спам, и распознанные ранее матрицы образов цифр.

На основе обучающей выборки строится модель (функция классификации или регрессии), которая определяет значение зависимой переменной. Чтобы получить наиболее точную функцию, необходимо удовлетворить несколько основных требований к обучающей выборке:

- количество объектов в выборке должно быть достаточным для обеспечения точности модели;

- выборка должна включать объекты из всех возможных классов в случае задачи классификации или всей области значений в случае регрессии;
- каждый класс или интервал значений в выборке должен содержать достаточное количество объектов.

Второй этап заключается в применении построенной модели к объектам с неопределенным значением зависимой переменной для получения результата.

Геометрия двумерного пространства может наглядно представить задачу классификации и регрессии. Представьте, что есть два независимых параметра, и каждое наблюдение представляет собой точку на плоскости. Символы «+» и «-» обозначают принадлежность объекта к одному из двух классов. Если мы видим четкую структуру данных -например, все точки одного класса («+») сосредоточены в центре, то задача классификации будет сведена к созданию поверхности или линии, которая обводит центральную область и назначает значение «+» для объектов внутри этой области и «–» для объектов за её пределами. Это позволяет классифицировать новые объекты, основываясь на их позиции относительно этой границы (рисунок 69).

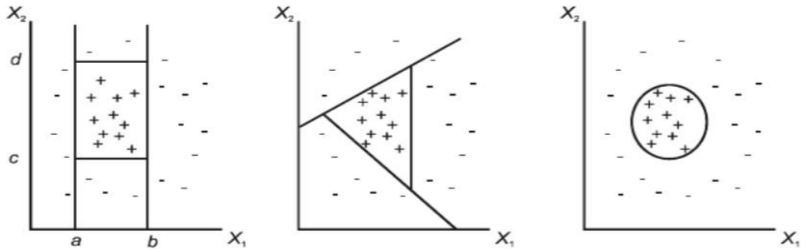

Рисунок 69. Классификация в двумерном пространстве

Из рисунка видно, что существует несколько способов построения обводящей области. Вид функции зависит от используемого алгоритма.

Основные проблемы, с которыми сталкиваются при решении задач классификации и регрессии, связаны с неудовлетворительным качеством исходных данных, которые могут содержать ошибочные данные и пропущенные значения. Также присутствуют различные типы атрибутов - числовые и категориальные, и разная значимость атрибутов. Ещё существуют так называемые проблемы overfitting и underfitting.

Проблема overfitting возникает, когда классификационная функция «слишком хорошо» адаптируется к данным. Она пытается интерпретировать ошибки и аномальные значения в данных как их

внутреннюю структуру. Это приводит к некорректной работе модели с другими данными, имеющие свой характер ошибок.

С другой стороны, проблема underfitting относится к ситуации, когда количество ошибок при проверке классификатора на обучающем множестве слишком велико. Это указывает на то, что-либо данные не имеют закономерностей, либо необходимо использовать другой метод для их обнаружения.

Кластеризация. Задача кластеризации заключается в разделении множества объектов на группы, которые называются кластерами. Слово «кластер» в переводе с английского языка означает «сгусток», «пучок» или «группа». Родственными понятиями являются класс, таксон и сгущение, а процедура разделения множества элементов на кластеры часто называется кластерным анализом.

Кластеризация находит применение в разнообразных областях, где необходимо провести исследование экспериментальных или статистических данных. Ярким примером такой задачи в маркетинге может служить сегментация.

Концепция сегментирования основана на различиях между потребителями в их потребностях, требованиях, поведении в процессе выбора, приобретения, использования товара и формирования отношения к нему. В связи с этим необходимо тщательно анализировать потребителей и предлагать им продукцию с разными характеристиками, продвигать и продавать товары по-своему. Основной целью сегментирования является установление, чем отличаются потребители друг от друга и как эти различия влияют на их требования к товару.

В маркетинге критериями сегментации являются географическое местоположение, социально-демографические характеристики, мотивы покупки и другие характеристики. Основываясь на результатах сегментации, маркетолог может определить ёмкость рынка, обнаружить потребности клиентов, которые не полностью удовлетворены существующими производителями, и принять решение о перспективности работы в каждом отдельном сегменте.

Для научных исследований кластеризация предоставляет возможность изучения причин, приводящих к объединению объектов в группы, что открывает новые направления в исследованиях. Классическим примером является периодическая таблица элементов, созданная Дмитрием Менделеевым в 1869 году. На тот момент было известно о существовании порядка 60 элементов, и они были разделены на кластеры. Изучение причин, по которым элементы разделялись на подобные группы, оказало

значительное влияние на приоритетные области исследований на будущее. Действительно, лишь через 50 лет квантовая физика предоставила убедительное объяснение периодичности таблицы.

Отличительной чертой кластеризации является отсутствие предварительной информации о зависимости между переменными, что делает её частью неконтролируемого обучения. Эта задача решается на начальных этапах исследования, когда о данных не существует достаточных знаний. Решение данной проблемы помогает лучше познать данные, и с этой точки зрения кластеризация служит описательной задачей.

Методы автоматического разбиения на кластеры редко используются самостоятельно, просто для получения групп схожих объектов. После определения кластеров применяются другие методы Data Mining для анализа последствий такого разделения и выявления его причин.

Кластерный анализ позволяет работать с большим объёмом информации, упрощает и делает её более наглядной. Суть процесса состоит в том, чтобы сжать массив данных, при этом сохранив его содержательность.

При проведении кластеризации необходимо учитывать несколько особенностей. Во-первых, метод зависит от природы изучаемых объектов (их характеристик). Они могут быть чётко определёнными и выраженными в количественных показателях, а могут -иметь вероятностное или нечёткое описание.

Во-вторых, кластеризация зависит от представления о самих кластерах и предполагаемых связях между объектами и кластерами. Необходимо определиться с возможностью принадлежности объекта сразу нескольким кластерам. Также нужно понять, что представляет собой принадлежность к кластеру для каждого конкретного объекта: принадлежность однозначную (объект либо принадлежит кластеру, либо нет), вероятностную (можно оценить вероятность принадлежности) или нечёткую (оценить степень принадлежности к кластеру).

Ассоциация. Поиск ассоциативных правил является очень востребованным методом Data Mining. Суть задачи состоит в выявлении часто встречающихся сочетаний объектов в большом наборе таких комбинаций, что представляет собой частный случай классификации. Эта задача изначально решалась через анализ потребительского поведения в супермаркетах, где изучались данные о товарах, помещаемых покупателями в тележку. Поэтому вторым названием метода является «анализ рыночных корзин» (Basket Analysis).

В ходе этого анализа наибольший интерес представляет информация о том, какие товары приобретаются вместе, какая категория потребителей предпочитает определённые товары, в какое время совершаются покупки и т. д. Эти сведения могут использоваться для более эффективного планирования закупок товаров, организации рекламных кампаний и др.

Например, благодаря анализу покупок можно сделать вывод, что, если покупаются чипсы или орехи, покупатели, как правило, берут пиво или воду соответственно. Обладая этими знаниями, можно размещать такие товары рядом друг с другом, объединять их в пакет со скидкой или предпринимать другие меры, чтобы стимулировать покупателя к покупке.

Аналогично применяются ассоциативные правила не только в торговле. В сфере услуг целесообразно выяснить, какими услугами клиенты предпочитают пользоваться совместно. Для получения этой информации задачу можно решить, применив её к данным об услугах, которые использует один клиент в течение определённого периода времени (например, месяца или года). Это помогает определить, как наиболее эффективно формировать пакеты услуг, предлагаемых клиенту.

Также эта методика применяется в медицине для исследования частоты сочетаний болезней и симптомов у пациентов. Эти знания полезны в будущем для правильной постановки диагноза.

Сиквенциальный анализ -это разновидность задачи поиска ассоциативных правил, при которой особый интерес представляет последовательность событий и обнаруживаются закономерности в таких последовательностях. Это позволяет с некоторой долей вероятности предсказывать появление событий в будущем.

Задача сиквенциального анализа отличается от простого поиска ассоциативных групп тем, что в ней устанавливается отношение порядка между исследуемыми наборами. Данное отношение можно определить разными способами. Например, если провести анализ последовательности событий, происходящих во времени, то объектами наборов будут сами события, а отношение порядка будет соответствовать хронологии их появления.

Данный вид анализа активно используется в телекоммуникационных компаниях -там с его помощью анализируют данные об авариях на различных узлах сети. Информация о последовательности аварий может помочь обнаружить неполадки и предотвратить возможные новые аварии. Важно учитывать порядок возникновения событий. К примеру, если известна последовательность сбоев:

{e5, e2, e7, e13, e6, e1, ...} (где ei обозначает сбой с кодом i),

можно сделать вывод, что, если произошёл сбой e2, стоит ожидать, сбоя e7. Зная это, можно предпринять профилактические меры, устраняющие причины возникновения сбоя. Иметь информацию о времени между сбоями также полезно -она может помочь не только предсказать факт появления, но также и время происшествия.

2. ***Краудсорсинг***. О краудсорсинге в современном его понимании стали говорить совсем недавно. Впервые определение краудсорсинга было предложено *Джеффом Хауи* и *Марком Робинсоном*, которые в 2006 году опубликовали статью под названием «The Rise of Crowdsourcing» в журнале Wired. В ней рассматривали краудсорсинг как инновационное социальное явление, возникающее в разнообразных сферах, как феномен объединения неограниченного количества людей для решения задач без какого-либо вознаграждения или за незначительную плату. Основной целью такого подхода является предложение задачи широкому кругу людей, чаще всего неограниченному, независимо от профессионализма, рода деятельности, уровня интеллектуального развития и возраста. Таким образом, множество людей узнали не только об инновационной технологии, но и о дешёвой рабочей силе, способной решить бизнес-задачи любого рода.

Многие компании проводят краудсорсинговые проекты, которые соответствуют типовому сценарию. Для успешного проведения проекта необходимо пройти шесть обязательных шагов. Однако компания может вносить дополнительные шаги на своё усмотрение, если считает это необходимым для решения поставленной задачи.

Эти шесть шагов были разработаны на основе анализа существующей отечественной и зарубежной литературы и интервью с экспертом в области краудсорсинга. Основные шаги:

1. Постановка задачи. На этом этапе заказчик должен определить цель, специфику, сроки выполнения проекта, группы участников (внутренние - сотрудники компании; внешние -клиенты, партнёры, интернет-сообщества и т. д.), формат итогового материала, условия проведения проекта, границы проекта, требования к уровню подготовки участников, награды и поощрения.

По итогам этапа устанавливаются правила, определяются цели и бюджет. Назначаются ответственные за каждый этап и человек, который будет нести ответственность за проект в целом.

2. Отбор участников.

3. Генерация идей. В рамках данного этапа участники проекта размещают свои идеи по проблеме на созданной платформе, и в большинстве случаев им доступна функция комментариев -у участников

есть возможность высказаться по той или иной идее и, возможно, внести в неё корректировки, которые помогут создать итоговое решение проекта.

Главным участником этого этапа является «фасилитатор» -тот, кто превращает процесс коммуникации в удобный для всех её участников. Понятие «фасилитатор» -калька с английского facilitator, производного от глагола to facilitate, который употребляется в английском языке с 1610-х годов и восходит к латинскому facilis. Оно означает «лёгкий, удобный».

На данном этапе главную роль играют фасилитаторы, которые: координируют участников и формируют общее видение цели проекта; создают формальные структуры как основу для общения людей с разными точками зрения и жизненным опытом; поддерживают активное взаимодействие участников для создания здоровой конкуренции и избежания конфликтов; сочетают индивидуальные высказывания с публичными ответами руководителя проекта на критические вопросы отдельных участников.

4. На данном этапе рабочая группа с экспертами занимается фильтрацией и отбором всех поступивших идей, решений и комментариев.

Сначала проходит голосование, чтобы выбрать наиболее значимые и ценные идеи. Затем эксперты формируют отношение к отобранным идеям: они определяют сильные и слабые стороны каждой концепции.

После этого выбираются идеи, сходные по смыслу и решению, на основе которых команда создаёт окончательное решение.

5. Отбор проекта (итог). Представляет собой процесс доработки решения, в котором участвуют не только эксперты рабочей группы, но и другие участники проекта. Результаты четвёртого этапа проекта размещаются на краудсорсинговой платформе, где участники находят наиболее ценные идеи и работают над ними сообща.

6. Подведение итогов. Заказчику предоставляются итоги проекта, он должен отобрать наиболее актуальные и подходящие идеи для внедрения.

Отбор идей осуществляется на основе следующих критериев:

- Поддержка самых ценных предложений, которые получили высшую оценку экспертов.
- Рейтинг авторов определяется по оценкам, которые набрали решения участников – от высшего к низшему.
- Учитываются количество опубликованных предложений, выставленных оценок и комментариев.
- Принимается во внимание количество участников и решений.

По итогам данного этапа заказчик получает список лучших решений, а также список лучших участников для вознаграждения.

Следуя этим шагам, компания может реализовать краудсорсинговый проект. Важно осознавать связь между всеми участниками проекта, поэтому автором была разработана схема взаимодействия на основе серий глубинных интервью с несколькими специалистами в области краудсорсинга.

3. ***Визуализация данных***. Визуализация данных -это представление данных в таком виде, который обеспечивает эффективное изучение этих данных человеком. Традиционно визуализация рассматривалась как вспомогательное средство при анализе данных, но теперь всё больше исследований говорит о её самостоятельной роли.

Визуализация является важной составляющей качественных систем анализа данных и может использоваться на всех этапах обработки данных.

С помощью визуализации данных можно решать следующие задачи:

- представить информацию наглядно;
- отобразить закономерности, характерные для исходного набора данных;
- сократить размерность данных;
- провести анализ качества данных (шумы, выбросы, пробелы);
- проиллюстрировать вид модели для анализа данных (структура нейронной сети и пр.);
- интерпретировать результаты, полученные в ходе анализа.

Разные типы данных нуждаются в различных способах их представления. Визуализация включает в себя такие этапы работы с данными, как оценка степени соответствия данных ожиданиям и пригодности данных к анализу, выдвижение гипотез о закономерностях и необходимых процедурах первичной обработки. Также сюда относится визуализация выборки, результатов первичной обработки, промежуточных и окончательных результатов. Метод визуализации – это системное, основанное на правилах, динамическое и/или статическое графическое представление информации, цель которого -помочь разобраться в сложных понятиях и дать обобщённое понимание.

В 2007 году *Ленглер* и *Эпплер* разработали периодическую таблицу, в которой классифицировали 100 различных способов визуализации данных. Таблица построена по тем же принципам, что и периодическая система химических элементов Дмитрия Менделеева.

Способы визуализации данных разделены в таблице на группы и периоды в зависимости от целей использования и сложности метода. Также они обозначены цветами, чтобы лучше показать тип визуализации. С

интерактивной таблицей можно ознакомиться на сайте *http://www.visual-literacy.org/periodic_table/periodic_table.html*.

При наведении курсора мыши на ячейку «всплывают» примеры для каждого метода визуализации. Некоторые из этих методов связаны с анализом данных и включают:

1) линейный график (line chart, area chart);
2) график рассеивания (scatterplot);
3) столбиковую диаграмму (bar chart);
4) гистограмму (histogram);
5) круговую диаграмму (pie chart);
6) площадную диаграмму (bubble chart);
7) древовидную структуру (tree structure);
8) диаграмму Венна–Эйлера (Venn/Euler diagram);
9) картограмму (cartogram);
10) дендрограмму (dendrogram).

Кроме средств визуализации данных, доступных во многих системах анализа, существуют специальные сервисы, направленные исключительно на визуализацию, такие как Many Eyes (*http://www.easel.ly*), Google Chart Tool (*https://developers.google.com/chart/*) и др.

Дополнительного внимания заслуживают средства визуализации, которые используются при анализе пространственных данных.

Пространственные данные -это сведения о географических объектах, об их местоположении и свойствах. Визуальное представление объектов демонстрирует их взаимное расположение и позволяет провести анализ разных пространственных отношений.

Цифровое описание таких объектов включает задание координат объекта наблюдения и описание его атрибутов.

Результаты анализа данных в картографических сервисах можно отображать различными способами. Например,

- Способ размерных символов (значков) означает, что анализируемые характеристики объектов отображаются специальными символами, размер которых передаёт количественную информацию, а форма и цвет - качественную.
- При способе качественного или количественного фона данные группируются по близости значений и созданным группам присваиваются определённые цвета, типы символов или линий.
- Точечный способ с изобразительным средством в виде множества точек одинакового размера позволяет отобразить количественный показатель. Каждая точка имеет своё значение этого показателя.

• Локализованные диаграммы позволяют показать соотношение нескольких характеристик, причём диаграммы привязаны к географии: например, в точке размещения поста наблюдений показано соотношение загрязняющих веществ.

5.2 Анализ экономической информации

А. Д. Шеремет определяет экономический анализ как средство получения цельного знания о хозяйственной деятельности, знаний о бизнесе, понимания деятельности экономического субъекта. Методологическую основу анализа составляют принципы материалистической диалектики и современного системного анализа, который в последние годы получил широкое распространение и в экономическом анализе.

Анализ -это метод научного исследования, процесс познания предметов и явлений окружающей среды. В узком плане он представляет собой разложение объекта на составные части с целью изучения их самостоятельного функционирования, а в широком -метод научного исследования. Он основан на расчленении целого на составные элементы и изучении их во всём многообразии связей и зависимостей.

Экономический анализ же представляет собой работу по разработке и совершенствованию методов и методик анализа, а также применению их для того, чтобы повысить эффективность управленческих решений.

Объектом экономического анализа является организация, которая в соответствии со ст. 11 ч. 1 Налогового кодекса Российской Федерации может рассматриваться как юридическое лицо, созданное в соответствии с законодательством РФ, либо с законами иностранных государств. Объектом также могут выступать международные организации, их филиалы и представительства, которые созданы на территории России.

Не только внутренняя, но и внешняя хозяйственная среда организации становится предметом рассмотрения в экономическом анализе, тогда как хозяйственная среда -это прочие организации и участники рынка, с которыми анализируемый объект взаимодействует в процессе своей деятельности. Потому, если сделать поправку на двойственность внешней и внутренней среды, объектом комплексно-экономического анализа становится организация вкупе со своей хозяйственной средой.

Предметом экономического анализа является информация о хозяйственной деятельности организации. Хозяйственной называют деятельность, связанную с производством и (или) продажей товаров,

выполнением работ или предоставлением услуг. В Российской Федерации функционируют сотни тысяч предприятий различных форм собственности, которые осуществляют свою деятельность в соответствии с действующим в стране законодательством. Они работают на принципах хозяйственной самостоятельности, материальной заинтересованности в результатах хозяйственно-финансовой деятельности и материальной ответственности за эти результаты. Также деятельность предприятий не может осуществляться бесконтрольно, без соответствующего анализа, поскольку в работе предприятий могут иметь место различные недостатки, ошибки и неиспользуемые резервы, негативно влияющие на эффективность производства.

Цель экономического анализа -повышение эффективности работы организации на основе системного изучения всех видов её деятельности и обобщения их результатов. Первая задача комплексного экономического анализа хозяйственной деятельности связана с оценкой качества, обоснованности и достоверности планов и нормативов. Следовательно, определение базовых показателей для планирования деятельности организации на предстоящий период также можно считать задачей экономического анализа. Две задачи тесно взаимосвязаны, так как удаётся выявить определённые экономические закономерности в развитии организации, именно благодаря анализу хозяйственной деятельности за предшествующие периоды проведения исследования.

Эти аспекты позволяют составить научно обоснованный план, в том числе учитывая основные факторы, которые оказывают существенное влияние на деятельность предприятия и должны быть учтены при составлении следующих планов.

Также после сбора необходимой информации можно провести анализ ожидаемого выполнения заданий за предшествующий период и внести уточнения в плановые показатели.

Третья задача экономического анализа подразумевает контроль за выполнением планов, а ещё оценку их исполнения. На этом этапе большую роль играют данные бухгалтерского и статистического учёта, сведения из других источников, например, отчёты. Тогда можно анализировать выполнение плана: как на текущий момент, так и за отчётный период, оценивая эффективность использования материальных, трудовых и финансовых ресурсов на предприятии.

Четвёртая задача -определение влияния отдельных факторов. При комплексном анализе важно выделить и измерить влияние внутренних (зависящих от деятельности предприятия) и внешних (отраслевых)

факторов. От этого будет зависеть правильность оценки деятельности организации и перспектив ее развития.

Пятая и последняя задача экономического анализа предполагает выявление резервов роста эффективности производства. Анализ действителен, когда приносит организации реальную пользу, которая заключается в выявлении неиспользованных возможностей на всех участках производства.

5.3 Система Business Intellegince (BI)

В условиях современного рынка для компаний крайне важно не только обеспечивать свою конкурентоспособность, но и повышать её. Прогнозирование и моделирование результатов деятельности компании могут стать эффективными инструментами для повышения конкурентоспособности. Этим требованиям соответствуют системы класса Business Intelligence (BI), которые представляют собой инструменты интеллектуального анализа данных.

Для компаний, работающих в сфере финансовых услуг и банковского сектора, где конкуренция особенно высока, а технологии развиваются стремительно, успех во многом зависит от способности компании управлять клиентским опытом и выстраивать долгосрочные отношения с клиентами. Банки, которые используют технологии Business Intelligence, могут повысить свою доходность и рентабельность, а также обеспечить прозрачность отчётности. Благодаря BI-решениям банк может подготовить для клиентов целевые предложения, что способствует расширению использования банковских продуктов, увеличению доходов банка и его лидерству на рынке.

В современном мире существует множество технологий анализа данных, но одной из самых новых и быстро развивающихся является технология интеллектуального анализа или бизнес–аналитики (Business Intelligence). Эта технология основана на организации доступа конечных пользователей к данным и анализе структурированных количественных данных и информации о бизнесе. Её применение поможет банковским структурам принимать обоснованные и эффективные управленческие решения на основе тщательно проанализированных данных.

Первое упоминание определения Business Intelligence зафиксировано американским учёным *Хансом Петером Луном*, специалистом в области information science (информационная наука). К появлению BI его подтолкнуло глубинное понимание сути информационных процессов и

роли информации в различных видах человеческой деятельности, в том числе и в бизнесе в современном его понимании.

До создания основ BI Ханс Петер Лун занимался статистическими исследованиями и индексацией текстов в докомпьютерную эпоху, когда были доступны только электромеханические табуляторы. Он известен своей разносторонностью: он проработал много лет в текстильной промышленности, сделал несколько изобретений, в том числе измерительный прибор «лунометр», который производится и используется по сей день. Однако в 50-е годы он изменил направление деятельности и начал активно заниматься разработкой методов работы с информацией; именно Лун предложил алгоритмы хеширования и полнотекстового поиска.

Перед тем как обозначить предмет своей работы, он определил его компоненты, описав business (бизнес) как набор различных активностей, предпринимаемых в науке, технологиях, коммерции, индустрии, законодательной деятельности, обороне и т.д. Коммуникационные системы, поддерживающие эти виды активности, он назвал intelligence system (системами, поддерживающими разумную деятельность).

А под intelligence (разумная деятельность) он понимал способность устанавливать взаимосвязь между представлениями отдельных фактов с тем, чтобы действовать в интересах решения поставленных задач и намеченных целей. Появление термина Business Intelligence датируется 1958 годом, когда американский учёный Ханс Петер Луна опубликовал в IBM System Journal статью «A Business Intelligence System» («Система бизнес-аналитики»).

Биосистемы в представлении Луна явно опередили время, поэтому в дальнейшем эта часть его работы была забыта на 30 лет, вплоть до 1989 года, когда её заново открыл известный аналитик из Gartner *Ховард Дреснер* и дал BI расширительную трактовку, предложив использовать BI (бизнес-аналитика) в качестве основного термина для различных технологий, предназначенных для поддержки принятия решений.

После этого начались расхождения во мнениях и поиск смысла BI. Дословно термин трактовался так: «ориентированный на пользователя процесс, который включает доступ и исследование информации, её анализ, выработку интуиции и понимания, которые ведут к улучшенному и неформальному принятию решений». Позже, в 1996 году, появилось уточнение: **Business Intelligence** - это инструменты для анализа данных, построения отчётов и запросов, которые могут помочь бизнес-пользователям. Идеи, высказанные *Дреснером* двадцать лет назад, теперь общеприняты. Поддержка бизнес-информации (BI) по-прежнему

рассматривается как комплекс слабо связанных между собой технологий. К ним относятся классические инструменты, такие как электронные таблицы, генераторы отчётов и технологии OLAP и OLAP soft. Также в эту область включаются средства для управления бизнес-процессами с цифровыми приборными щитами, технологии разработки данных и текстов и многое другое.

Эти инструменты различаются по сложности, но их объединяет общая цель, определённая Луном - помогать человеку в превращении данных в полезную для него информацию.

Каждый сотрудник и руководитель сталкиваются с проблемами составления и анализа отчётности. Обычно это огромные базы данных в виде плоской таблицы Excel, работать с которой может только один пользователь. Такие таблицы раскрываются только в одном разрезе, и если в следующем месяце руководитель решит проанализировать отчёт, например, в разрезе банковских продуктов или по сделкам, а не по сотрудникам, то необходимо будет перестраивать структуру.

Эти проблемы требуют внедрения аналитической системы, которая могла бы вести как учёт данных, так и анализировать эти данные и принимать решения на основе этих анализов. Она должна позволять отфильтровывать данные, разворачивать их в разных разрезах и давать возможность взглянуть на данные под разным углом.

Бизнес-аналитика (BI) - это аналитические системы, объединяющие данные из различных источников, обрабатывающие их и предоставляющие удобный интерфейс для всестороннего изучения, и оценки полученных сведений. Такие данные могут достигать поставленных бизнес-целей благодаря оптимальному использованию имеющихся данных. Комплексный анализ данных по всем направлениям бизнеса позволяет повысить его эффективность и снизить издержки.

В соответствии с подходом аналитиков Gartner Group выделяют три основных типа инструментальных средств BI (рисунок 70).

Средства создания отчетов (*Reporting*) позволяют создавать форматированные интерактивные отчеты. Это очень важная часть BI-платформ, поэтому они должны предоставлять пользователям широкий набор типов отчетов, таких как финансовые или операционные, в виде информационных панелей показателей (*Dashboards*).

Информационные панели показателей (*Dashboards*) -одна из составляющих отчетов. Они представляют информацию в интуитивно понятном графическом изображении, включающем диаграммы, круговые шкалы, светофоры и тому подобное. Применение таких индикаторов

помогает отслеживать состояние того или иного анализируемого параметра по сравнению с его целевым назначением.

Функция под названием «генератор нерегламентированных запросов» (Ad hoc query), которую еще называют созданием отчетов в режиме самообслуживания, дает пользователям возможность получать ответы на интересующие их вопросы. Система предоставляет пользователям средства навигации по доступным ресурсам данных.

Рисунок 70. Инструментальные средства BI

В некоторых случаях BI–платформы используются как промежуточное звено в цепочке анализа информации. При этом Microsoft Office, а именно, Excel, выступает как BI–клиент. В таких случаях крайне важно, чтобы BI–вендор обеспечил интеграцию с Microsoft Office, включая поддержку форматов документов, формул и сводных таблиц.

Все инструменты BI-платформы должны использовать одни и те же инструменты для обеспечения безопасности, общие метаданные и администрирование, средства генерации запросов и иметь однотипные интерфейсы. Необходима единая инфраструктура для полноценной работы платформы.

Кроме этого, все инструменты приложения должны опираться на единые метаданные. Важно обеспечить быстрый поиск, хранение, использование и публикацию таких объектов метаданных, как размерности, иерархии, параметры оценки производительности и оформления отчетов.

Метаданные -это данные о данных. Они предоставляют пользователям информацию о характере данных, об источнике их происхождения и способах доступа к ним.

Контрольные вопросы по главе 5

1. Дайте определение понятию «Dig Data».
2. Перечислите характеристики больших данных.
3. Что включает в себя описательная модель больших данных?
4. Охарактеризуйте интеллектуальный анализ данных.
5. Какие задачи интеллектуального анализа вы знаете? Опишите каждый.
6. Охарактеризуйте краудсорсинг.
7. Охарактеризуйте визуализацию данных.
8. Перечислите задачи экономического анализа.
9. Охарактеризуйте системы Business Intellegince.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аврунев А.В. Модели баз данных: учебное пособие / О.Е. Аврунев, В.М. Стасышин. Новосибирск: Изд-во НГТУ, 2018. 124 с.
2. Алетдинова А.А. Интеллектуальный анализ больших данных: учебное пособие / А.А. Алетдинова, М.Ш. Муртазина. Новосибирск: Изд-во НГТУ, 2023. 66 с.
3. Воронина В.В. Теория и практика машинного обучения: учебное пособие / В.В. Воронина, А.В. Михеев, Н.Г. Ярушкина, К.В. Святов. Ульяновск: УлГТУ, 2017. 290 с.
4. Григорьев Ю.А., Плутенко А.Д., Плужникова О.Ю. Реляционные базы данных и системы NoSQL: учебное пособие. Благовещенск: АмГУ, 2018. 424 с.
5. Карташева О.В. Современные информационные технологии в экономике и управлении: учебное пособие. М.: Прометей, 2024. 100 с.
6. Кваша В.А., Воеводина Е.И., Юрченко А.В., Бурыкин А.Д., Ларионов В.Д. Классификация методов и алгоритмов мягких вычислений // Мягкие измерения и вычисления. 2024. № 4. С. 5–14.
7. Кулаков К.А. Технологии XML в 2 ч.: учебное пособие для студентов математического факультета / К.А. Кулаков, В.М. Димитров. Петрозаводск: Изд-во ПетрГУ, 2014. 68 с.
8. Любанович Билл. Простой Python. Современный стиль программирования. СПб: Питер, 2021. 592 с.
9. Розина А.С., Воеводина Е.И., Колесов Р.В., Сироткин С.А., Хованская П.П. Использование методов мягких вычислений для прогнозирования денежных потоков // Мягкие измерения и вычисления. 2024. № 5. С. 74–82.
10. Рындина С.В. Базовые возможности языка Python для анализа данных: учеб. -метод. пособие. Пенза: Изд-во ПГУ, 2022. 72 с.
11. Тихонов Д.В., Ермоленко М.О., Русинова О.Ю. Информационные технологии в цифровой экономике: учебное пособие. Ярославль: ООО «ПКФ «СОЮЗ-ПРЕСС», 2024. 144 с.

Учебное пособие

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ И ОБРАБОТКИ ДАННЫХ

ISBN 978-5-6048031-9-6

Подписано в печать 03.06.2024. Формат 60x90/16.
Усл. печ. л. 14,5. Тираж 30 экз. Заказ № 3171.

Отпечатано в ООО «ПКФ «СОЮЗ-ПРЕСС»
150062, г. Ярославль, пр-д Доброхотова, д.16, кв.158
Тел.: (4852) 58-76-33, 58-76-37