МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Л.М. Олещенко

ТЕХНОЛОГІЇ ОБРОБЛЕННЯ ВЕЛИКИХ ДАНИХ

Конспект лекцій

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів, які навчаються
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення мультимедійних та
інформаційно-пошукових систем»)

Київ
КПІ ім. Ігоря Сікорського
2021

Рецензенти: Бідюк Петро Іванович, д-р техн. наук, проф.
Полторак Вадим Петрович, канд. техн. наук, доц.

Відповідальний редактор Легеза Віктор Петрович, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 8 від 24.06.2021 р.)
за поданням Вченої ради факультету прикладної математики
(протокол № 12 від 31.05.2021 р.)

Електронне мережне навчальне видання

Олещенко Любов Михайлівна, канд. техн. наук, доцент

ТЕХНОЛОГІЇ ОБРОБЛЕННЯ ВЕЛИКИХ ДАНИХ

КОНСПЕКТ ЛЕКЦІЙ

Технології оброблення великих даних: конспект лекцій з дисципліни «Технології оброблення великих даних» [Електронний ресурс] : навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»)/ Л.М. Олещенко; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 5,55 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 227 с.

Навчальний посібник розроблено для вивчення студентами теоретичних відомостей з технологій оброблення великих даних. Навчальне видання призначене для студентів, які навчаються за спеціальністю 121 Інженерія програмного забезпечення (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем») факультету прикладної математики КПІ ім. Ігоря Сікорського.

ЗМІСТ

ВСТУП

Аналітика великих даних сьогодні є однією з найбільш затребуваних у сучасному бізнесі. Знання нових технологій програмування та уміння розробляти програмне забезпечення для управління та аналізу великих об'ємів інформації використовуються для забезпечення цифровізації усіх сфер життя.

Дисципліна «Технології оброблення великих даних» входить до циклу професійно-орієнтованих дисциплін плану підготовки бакалаврів, що навчаються за спеціальністю 121 «Інженерія програмного забезпечення» (освітня програма «Програмне забезпечення мультимедійних та інформаційно-пошукових систем»).

Предметом дисципліни є теоретичні та практичні основи оброблення великих даних. Розглядаються загальні методи оброблення великих даних, можливості мов програмування Python та R для аналізу та візуалізації даних.

Мета дисципліни – забезпечити знання теоретичних і практичних основ з оброблення великих даних за допомогою мов програмування Python, R та технологій розподіленого оброблення даних.

Метою конспекту лекцій є отримання необхідного рівня знань технологій програмування для оброблення великих даних, архітектурних моделей Big Data, технологій віртуалізації, контейнерних технологій виконання програмного коду на сервері. У даному конспекті лекцій студенти ознайомляться з основними джерелами даних та технологіями оброблення великих даних з використанням інструментів Python та R.

Курс лекцій з дисципліни «Технології оброблення великих даних» розрахований на 36 академічних годин аудиторних занять. Конспект лекцій складається з 18 розділів, кожен з яких присвячений одній лекції з дисципліни «Технології оброблення великих даних». У кожному розділі надаються теоретичні відомості з кожної теми, контрольні питання для самоперевірки та список рекомендованої літератури.

Лекція 1.
Джерела великих даних. Інтернет Речей.
Визначення Big Data

План лекції

1.1. Інтернет Речей та зростання даних.
1.2. Платформа Kaggle.
1.3. DrivenData.
1.4. Визначення великих даних.
1.5. Приклади великих даних у реальному світі.
1.6. Відкриті дані.
1.7. Приватність даних.
1.8. Структуровані та неструктуровані дані.
1.9. Хмарні та туманні обчислення.
1.10. Дані в спокої та дані в русі.
1.11. Інфраструктура великих даних.
1.12. Розподілені дані та їх обробка.

1.1. Інтернет Речей та зростання даних

Наш світ є дуже складним. Ця складність зумовлює генерування постійно зростаючого обсягу даних, які потрібно зберігати та аналізувати. Швидкість генерування даних не виявляє ознак уповільнення. Різноманітність даних поширюється на нові області, які ніколи не були доступні для аналізу. Взаємодія між людьми, що використовують медіаплатформи, автоматизація процесів та агрегація даних, що надходять з різних джерел, створюють Інтернет речей. Інтернет речей (Internet of Things, IoT) не тільки приєднує датчики до існуючих речей, він створює ринок нових пов'язаних речей. Усі ці пов'язані речі генерують дані. Це додає немислиму кількість великих даних, що називається Big Data.

Збирати всі наявні дані в рамках проекту чи рішення не завжди можливо. Кількість даних, яку можна зібрати, визначається можливостями датчиків, мережі, комп'ютерів та іншого обладнання. Це також визначається необхідністю, наприклад, для перевірки правильності вирівнювання етикетки кожної пляшки, що рухається високошвидкісною лінією розливу напою У цьому випадку важливі дані кожної пляшки. Для іншого датчика, такого як датчик вологості в кукурудзяному полі, не потрібно повідомляти про вимірювану величину вологості кожну десяту секунду. Кожні п'ять-десять хвилин може бути

достатньо. Не всі зібрані дані можна використовувати як є у первинному вигляді. Можливо, були зібрані сторонні, невірні або неправдиві дані. Щоб зробити ці дані корисними, їх потрібно очистити. Очищення – це видалення небажаних даних, зміни неправильних даних та заповнення відсутніх даних. Для очищення даних прийнято використовувати програмний код. Це досягається шляхом пошуку критеріїв або їх відсутності та оперування даними, поки не буде більше аномалій. Після очищення даних їх можна легше шукати, аналізувати та візуалізувати. За допомогою аналізу даних можна дізнатись цікаві відомості та виявити тенденції. Це часто призводить до нових запитів, які ще не були реалізовані. Якщо не можна виявити додаткову цінність з деякого набору даних, можна експериментувати з тим, як вони організовані та представлені. Наприклад, камера безпеки, яка стежить за парковкою для злочинів, може також використовуватися для повідомлення водіям про кількість та розташування вільних місць. Якщо вважати, що кожне зерно рису еквівалентне одному байту даних, то при послідовному подвоєнні зерен в кожній наступній комірці кількість рисових зерен в останньому квадраті шахової дошки буде еквівалентна дев'яти екзабайт, як показано на рис.1.1. Один екзабайт становить приблизно 1,07 мільярда гігабайт. Дев'ять екзабайтів приблизно еквівалентні обсягу інтернет-трафіку за 2014 рік [1].

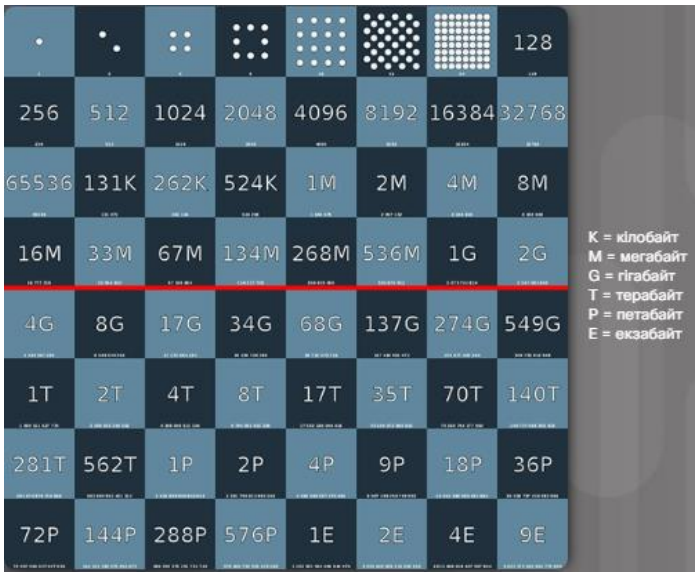
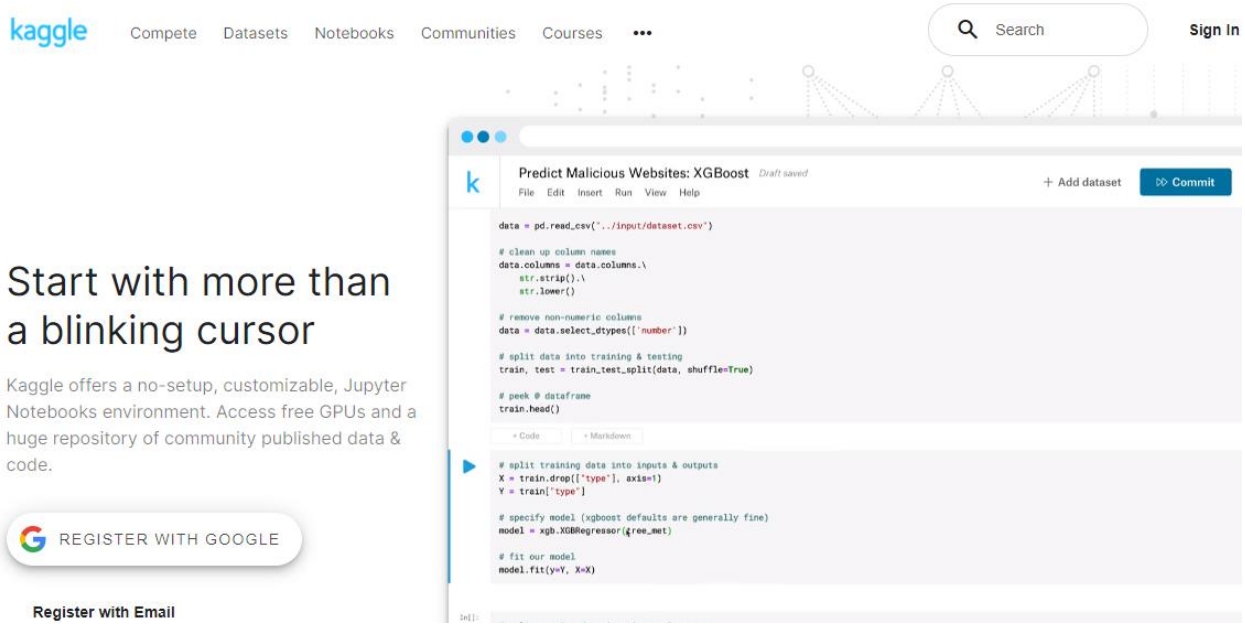


Рис.1.1. Подвоєння кількості байтів в комірках шахової дошки [2]

1.2. Платформа Kaggle

Інновації дозволяють компаніям не зупинятися в розвитку. Сьогодні все більше організацій розміщують датчики у свої продукти. Їх мета – збір та аналіз даних для отримання цінних відомостей. Щоб використовувати можливості IoT, організації потребують кваліфікованих та творчих людей. Інтернет-платформи, такі як Kaggle, дозволяють компаніям знаходити талановитих людей з різних куточків світу.

Kaggle – це платформа, яка об'єднує підприємства та організації, що мають питання щодо своїх даних та людей, які знають, як знайти відповіді на ці питання. Різні організації проводять змагання онлайн для створення кращих світових моделей даних. Учасники змаганнях генерують багато моделей, використовуючи різноманітні прийоми. Гравці з усього світу мають різну освіту та спеціалізації. Вони можуть підключитися до команд або просто допомогти один одному. Переможець або команда-переможець кожного змагання виграє приз. Зазвичай це можуть бути гроші, але іноді це може бути запрошення на роботу у відповідну компанію. У кожному змаганні учасники постійно вдосконалюються, оскільки кожен переможець долає попередній бал. Нові прогнозні моделі даних постійно перевершують існуючі кращі моделі. Клініка Майо, NASA, GE та Deloitte – це лише декілька підприємств та організацій, які приймали змагання з Kaggle [3].



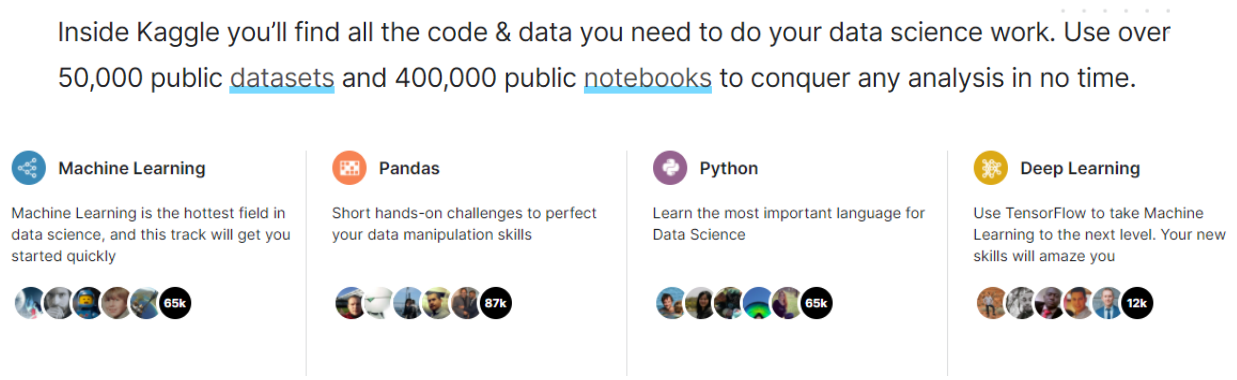


Рис.1.2. Платформа Kaggle [3]

1.3. DrivenData

Технології, що застосовуються в IoT та аналітиці даних, можуть бути використані для вирішення різних соціальних проблем. Зібрані дані можна використовувати для прогнозування різноманітних тенденцій. Наприклад, використовуючи доступні дані, підприємці в країні можуть передбачити, які водяні насоси функціонують, а які потребують ремонту чи не працюють. Завдяки прогнозуванню робота пристроїв обслуговування стає більш ефективною. Змагання, які виконуються для різних соціальних проектів, наприклад, можна знайти на веб-сайті DrivenData [4].

Місією DrivenData є використання передових практик з аналізу даних та краудсорсингу для вирішення глобальних соціальних проблем. Як і Kaggle, вони приймають виклики в Інтернеті, глобальна спільнота науковців змагається за створення найкращої статистичної моделі для складних проблем прогнозування. Ці моделі можуть сприяти позитивним змінам у світі.

DrivenData починається з постановки прогнозного питання, яке може бути вирішене за наявними даними та мати вимірюваний, реальний вплив. Вони співпрацюють з некомерційними організаціями, щоб зрозуміти їхні потреби та виявити продуктивні партнерські стосунки. Далі DrivenData проводить онлайн-конкурс з відкритими інноваціями, де розробники програм та науковці даних подають статистичні моделі. Використовуючи свою конкурентну платформу та механізм оцінювання, моделі класифікуються залежно від того, наскільки добре

вони прогнозують дані конкурентів. І нарешті, вони працюють з організацією, щоб використовувати найкращу модель, новий статистичний підхід або інструмент для аналізу даних. Це дає можливість неприбутковій організації більш ефективно та стабільно виконувати свою місію.

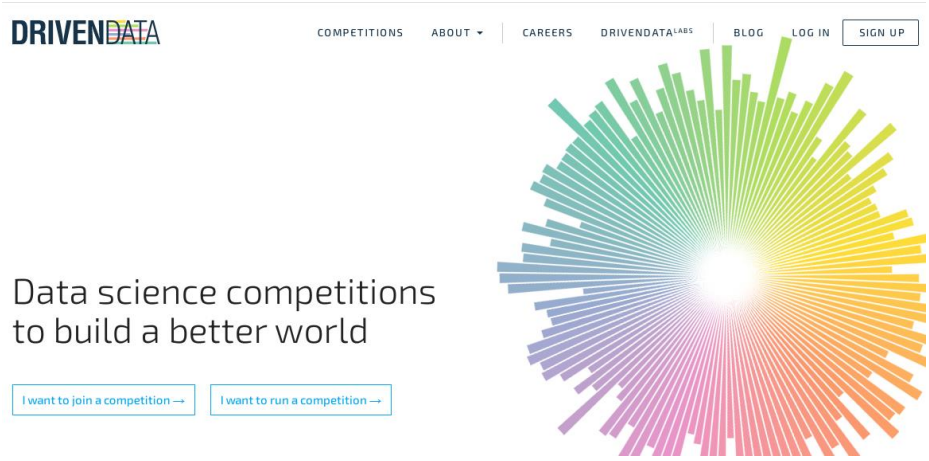


Рис.1.3. Платформа DrivenData [4]

1.4. Визначення великих даних

Експонентне зростання даних створило нову сферу інтересів у галузі технологій та бізнесу під назвою "Big Data". Загалом, набір даних або бізнес-проблема належать до класифікації Big Data, коли її дані настільки великі або складні, що їх стає неможливим зберігати, обробляти та аналізувати, використовуючи традиційні підходи до зберігання та аналізу даних.

Скільки даних потрібно, щоб стати Big Data? Чи достатньо 100 терабайт або 1000 петабайт? Обсяг є лише одним із критеріїв, оскільки потреба в обробці даних у режимі реального часу (також це називають даними в русі) або потреба в інтеграції структурованих і неструктурованих даних може кваліфікувати проблему як велику проблему даних. Наприклад, Міжнародна корпорація даних (International Data Corporation, IDC) використовує 100 терабайт як розмір набору даних, який визначається як Big Data. Якщо дані потокові, розмір набору даних може бути меншим, ніж 100 терабайт, але все ще вважається Big Data до тих пір, поки дані, що створюються, збільшуються на понад 60% на рік. Згідно National Institute of Standards and Technology (NIST): "Парадигма великих даних

складається з розподілу систем даних по горизонтально пов'язаних незалежних ресурсах для досягнення масштабованості, необхідної для ефективної обробки великих наборів даних".

Щоб вирізнити дані від великих даних, використовуються чотири Vs:

- **Об'єм (Volume)** - кількість даних, що передаються та зберігаються. Поточним завданням є пошук способів найбільш ефективно обробити зростаючий обсяг даних.
- **Швидкість (Velocity)** - швидкість, з якою формуються дані. Наприклад, дані, згенеровані мільярдом акцій, проданих на Нью-Йоркській фондовій біржі, не можуть бути просто збережені для подальшого аналізу.
- **Різноманітність (Variety)** - тип даних, який рідко знаходиться в стані, який ідеально готовий до обробки та аналізу. Велика частка Big Data – неструктуровані дані, які, за оцінками, становлять від 70 до 90% світових даних.
- **Достовірність (Veracity)** - процес запобігання неточного опису наборів даних. Наприклад, люди можуть створювати онлайн-акаунт та використовувати неправдиву контактну інформацію. Підвищена правдивість у зборі даних зменшує необхідну кількість очищення даних.

Хоча тут перераховано чотири V, більшість дискусій, інструментів та документів стосуються лише перших трьох (об'єм, швидкість, різноманітність).

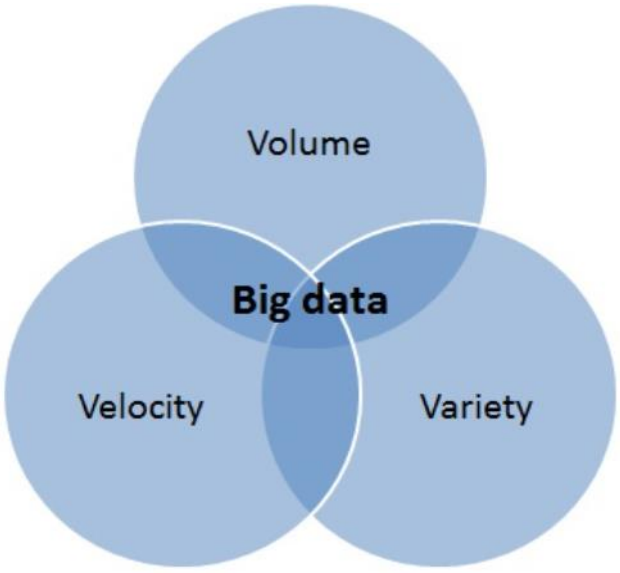


Рис.1.4. Характеристики Big Data [5]

1.5. Приклади великих даних у реальному світі

Розглянемо кілька прикладів у реальному світі генераторів великих даних. Airbus A380 Engine генерує 1 петабайт даних під час рейсу з Лондона в Сінгапур. Великий адронний колайдер (Large Hadron Collider, LHC) генерує 1 гігабайт даних щосекунди. Квадратний кілометровий масив (Square Kilometre Array, SKA) є найбільшим радіотелескопом у світі. Він генерує 20 екзабайтів даних на день. Це еквівалентно 20 мільярдам гігабайт на день [6].

Інтернет речей використовує датчики для створення даних. Дані можуть надходити від датчиків температури та вологості, які є у сільському господарстві. Датчики зараз є у всьому, від смартфонів до автомобілів та реактивних двигунів до побутової техніки. Список речей з датчиками зростає з кожним роком, це також сприяє експоненційному зростанню Big Data.

1.6. Відкриті дані

З підвищенням важливості даних для бізнесу та людей виникає багато питань щодо конфіденційності та наявності великих сховищ публічних та приватних даних. Для аналітиків важливо розуміти континуум між відкритими та приватними даними. Прийняття рішень про те, як будуть використовуватися різні типи даних в організації, так само важливі, як і знання про те, як реалізувати розподілене зберігання та оброблення Big Data.

Фонд "Open Knowledge Foundation" [7] визначає відкриті знання як "будь-який вміст, інформацію або дані, які люди можуть використовувати і перерозподіляти без будь-яких юридичних, технологічних чи соціальних обмежень". Відкриті дані є складовими відкритих знань. Відкриті знання – це коли відкриті дані стають корисними та використовуються .

Цінність відкритих даних можна відразу побачити, переглянувши такі сайти, як Портал відкритих даних Нью-Йорка, відкриті дані NYC, де відвідувач може швидко знайти рейтинги ресторанів на основі щорічних перевірок Міністерства охорони здоров'я та психічної гігієни. Портал є посередництвом із понад 1300

наборів даних від міських агентств для сприяння прозорості уряду та громадської активності. Набір даних – це сукупність пов’язаних дискретних записів, які можуть бути доступні для управління окремо або як ціле об'єднання.

Gapminder – некомерційне підприємство, що сприяє сталому глобальному розвитку [8]. На сайті представлена аналітика відкритих наборів даних із уточненням статистичних даних на такі теми, як:

- здоров'я та багатство націй;
- викиди CO_2 з 1820 року;
- дитяча смертність;
- ВІЛ-інфекція.

Портал відкритих даних України [9] містить набори даних за групами Будівництво, Держава, Екологія, Економіка та бізнес, Земля, Молодь і спорт, Освіта і культура, Охорона здоров'я, Податки, Сільське господарство, Соціальний захист, Стандарти, Транспорт, Фінанси, Юстиція у таких форматах, як csv, xls, JSON.

1.7. Приватність даних

По мірі розроблення нових програмних додатків від кінцевого користувача вимагається все більше даних, щоб дати компаніям та рекламодавцям більше інформації для прийняття бізнес-рішень. Використовуючи SafeAnswers, openPDS надає лише відповіді на конкретні запити, а необроблені дані не надсилаються. Розрахунок для відповіді здійснюється в сховищі персональних даних користувача (personal data store, PDS): "Тільки відповіді, узагальнені дані, необхідні додатку, залишають межі PDS користувача (наприклад, експорт GPS даних для додатка, щоб дізнатися, чи користувач активний або, дізнатися про загальну географічну зону, обчислення може бути зроблено в PDS користувача відповідної Q & A модуля". Конфіденційність даних користувачів спеціалісти почали обговорювати в 90-х роках XX століття, сьогодні просувається думка про те, що майбутнє конфіденційності не може бути забезпечене виключно

дотриманням законодавства та нормативно-правової бази; швидше, забезпечення конфіденційності має стати режимом роботи організації за замовчуванням.

1.8. Структуровані та неструктуровані дані

Раніше ми класифікували дані як відкриті або приватні з точки зору їх доступності. Дані також можна класифікувати за способом їх упорядкування, як структуровані або неструктуровані.

Структуровані дані вводяться та підтримуються у фіксованих полях у файлі чи записі. Структуровані дані ми можемо легко вводити, класифікувати, робити запити та аналізувати комп'ютером. Сюди входять дані, знайдені у реляційних базах даних та електронних таблицях.

Якщо набір даних досить малий, для структурованих даних використовують структуризовану мову запитів (Structured Query Language, SQL), мови програмування, створеної для запиту даних у реляційних базах даних. SQL працює лише на структурованих наборах даних. Однак для Big Data структуровані дані можуть бути частиною набору даних, але інструменти Big Data не залежать від цієї структури. Big Data переважно має набори даних, які складаються з неструктурованих даних.

Рис. 1.5. Структуровані дані [2]

Неструктуровані дані – це необроблені дані, вони не впорядковані заздалегідь і не мають фіксованої схеми, яка б ідентифікувала тип даних. У неструктурованих даних бракує встановленого способу введення або групування даних та їх аналізу. Приклади неструктурованих даних включають вміст фотографій, аудіо,

відео, веб-сторінок, блогів, книг, журналів, дописів, презентацій PowerPoint, статей, електронної пошти, вікі-файлів, текстових текстових документів та тексту в цілому. Прикладом неструктурованих даних є PDF-версія книги. Текст можна шукати, але він не організований у заздалегідь визначеній формі, наприклад, за допомогою полів та записів. Як структуровані, так і неструктуровані дані є цінними для людей, організацій, галузей та урядів. Організаціям важливо прийняти всі форми даних та визначити способи їх форматування, щоб ними можна було керувати та аналізувати.

1.9. Хмарні та туманні обчислення

У минулому набори даних були в основному статичними, розташовувались на одному сервері або в колекції серверів усередині організації та оброблялися з використанням мови програмування бази даних, такої як SQL. Хоча ця модель усе ще існує, зберігання великих наборів даних перемістилося в центри обробки даних (ЦОД). Сьогодні, із зростанням хмарних обчислень, ролі Big Data та необхідністю аналізу даних у режимі реального часу, дані продовжують зберігатись у ЦОД. Дані також повинні бути доступними для аналізу ближче до місця їх створення, і знання, отримані на основі цих даних, можуть мати найбільший вплив. Такий спосіб оброблення даних називається туманним обчисленням (Fog computing).

Туман – це хмара, розташована близько до джерела генерації даних. Туманні обчислення не є заміною хмарних обчислень, скоріше, туманні обчислення дозволяють розробляти нові інструменти. У моделі обчислень туману існує взаємозв'язок між хмарою та туманом, особливо коли мова йде про управління даними та аналітику. Туманні обчислення забезпечують обчислення, зберігання та мережеві послуги між кінцевими пристроями та традиційними ЦОД. Туманні обчислення виробляють величезну кількість даних від різних датчиків і контролерів. При роботі з даними в Інтернеті необхідно враховувати три важливі фактори:

- **Енергія або акумулятор** – кількість енергії, яка використовується датчиком IoT та залежить від швидкості вибірки датчика. Діапазон між пристроями також може впливати на кількість енергії, яка повинна бути використана для передачі даних сенсорів контролерам. Чим далі датчик, тим більше енергії потрібно використовувати для передачі даних.
- **Ширина смуги пропускання** – коли багато датчиків передають дані, може виникнути затримка зв'язку, якщо недостатньо пропускної здатності для підтримки усіх пристроїв. Додатковий аналіз в тумані може допомогти зменшити деякі вимоги до смуги зв'язку.
- **Затримка** – на аналіз даних у режимі реального часу впливає занадто велика затримка в мережі. Дуже важливо, щоб виконувались лише необхідні комунікації з хмарою, і обчислення відбувалося якомога ближче до джерела даних.

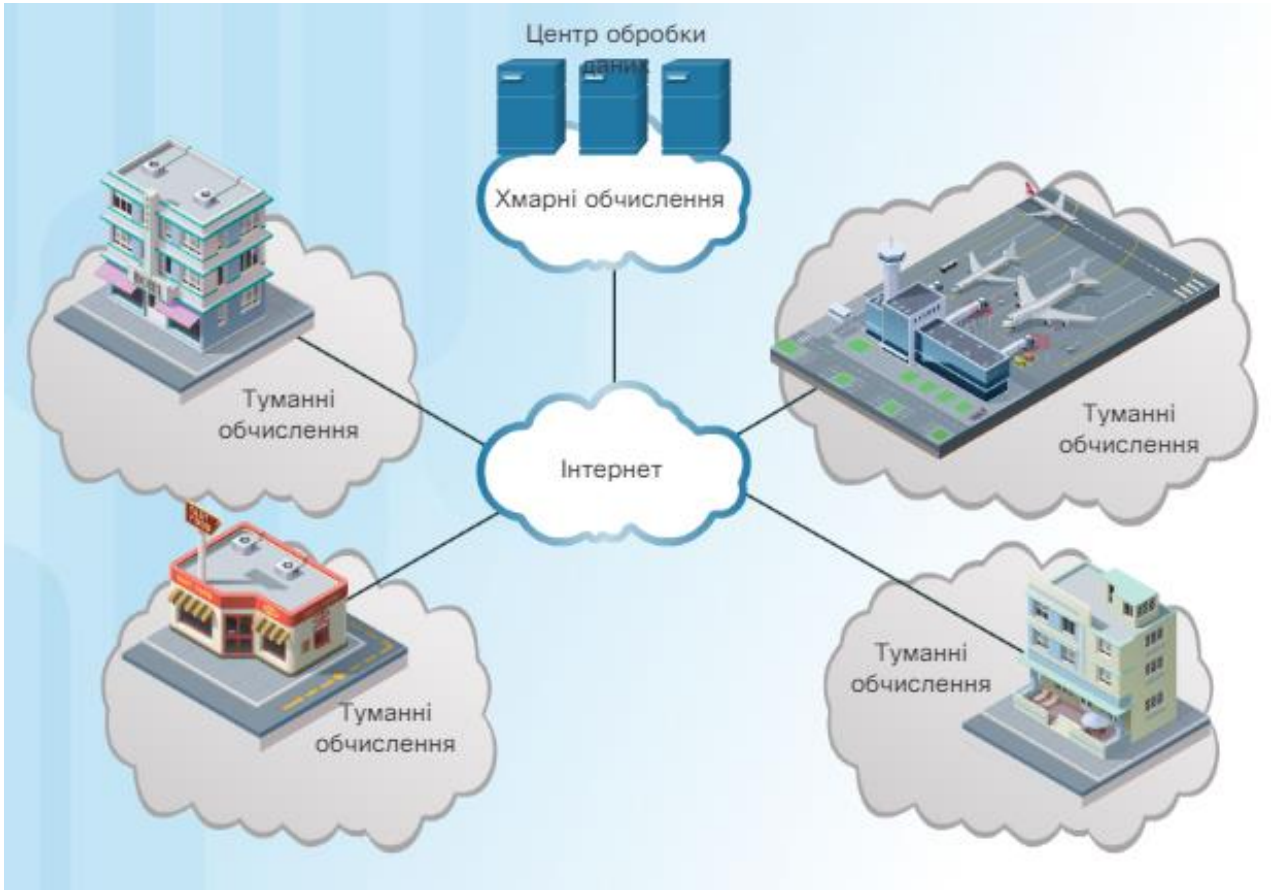


Рис. 1.6. Модель туманних обчислень [2]

1.10. Дані в спокої та дані в русі

Дані в спокої – це статичні дані, які зберігаються у фізичному місці, наприклад, на жорсткому диску на сервері чи в ЦОД. Дані зберігаються в базі даних, а потім аналізуються та інтерпретуються. Ті, хто приймає рішення, отримують повідомлення про те, чи потрібно діяти.

Дані, що перебувають у русі – це динамічні дані, які потребують обробки в режимі реального часу, перш ніж ці дані стають неактуальними або застарілими. Це безперервна взаємодія між людьми, процасами та речами. Пристрої на краю мережі працюють разом, щоб негайно діяти на знаннях, отриманих завдяки динамічному аналізу даних. Порядок аналізу, дії та повідомлення може бути різним. Важлива відмінність даних у стані спокою і даних у русі полягає в тому, що при даних, що перебувають у русі, дія даних на них відбувається до зберігання даних.

Дані, що перебувають у русі, використовуються різними галузями, які покладаються на отримання значень із даних перед їх збереженням. Датчики в полі фермера постійно надсилають дані про температуру, вологість ґрунту та сонячне світло до місцевого контролера, який аналізує дані. Якщо умови не правильні, контролер діє негайно, надсилаючи сигнали виконавчим механізмам у полі, щоб розпочати полив. Потім контролер надсилає повідомлення власнику поля про початок поливу та надсилає дані для зберігання.

Через особливості Big Data неможливо дублювати та зберігати всі ці дані у централізованому сховищі даних. Нові реалізації пристроїв включають велику кількість датчиків, що фіксують та обробляють дані. Рішення та дії потрібно проводити на межі, де і коли створюються дані. Оскільки датчики набирають більше процесорної потужності та стають більш усвідомленими у контексті, тепер можна наблизити інтелектуальні та аналітичні алгоритми до джерела даних. У цьому випадку дані, що знаходяться в русі, залишаються там, де вони створені, і представляють уявлення в реальному часі, спонукаючи до кращих, швидших рішень.

1.11. Інфраструктура великих даних

Багато компаній розуміють, що є сенс інвестувати в технології Big Data, щоб залишатися конкурентоспроможними на своєму ринку. На даний час їх інфраструктура даних може виглядати приблизно так, як рис.1.7, із серверами баз даних та традиційними засобами обробки даних. Зазвичай доступ до даних обмежений кількома відповідальними особами в організації. Компанії швидко рухаються до використання технологій Big Data для керування бізнес-аналітикою. За даними National Institute of Standards and Technology (NIST), парадигма Big Data складається з розподілу систем даних по горизонтально пов'язаних незалежних ресурсах для досягнення масштабованості, необхідної для ефективної обробки великих наборів даних. Це горизонтальна масштабованість. Він відрізняється від вертикальної масштабованості тим, що не намагається додати більше процесорної потужності та пам’яті існуючим машинам. Ці інфраструктури дозволяють багатьом користувачам одночасно безперешкодно та безпечно отримувати доступ до даних. Одним із таких прикладів є тисячі інтернет-покупців або мобільних геймерів.

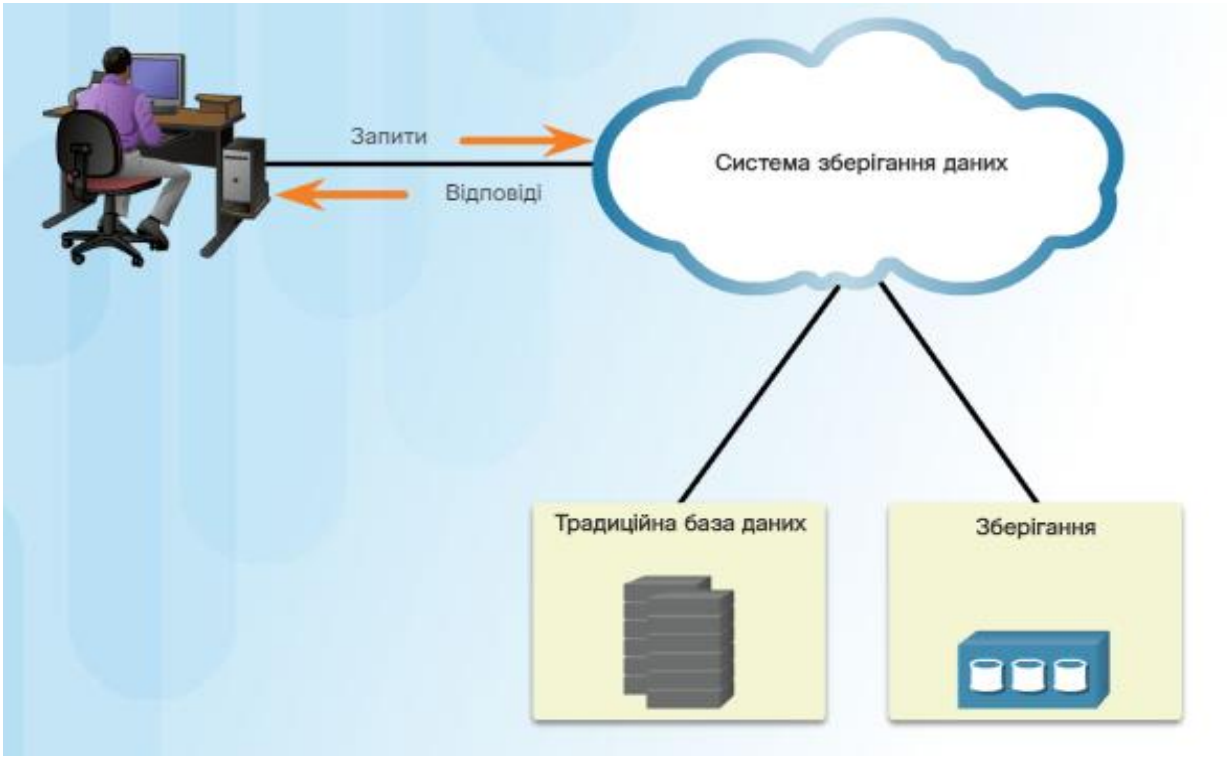


Рис. 1.7. Традиційна система управління базами даних [2]

На рис. 1.8 представлено пристрої в інфраструктурі великих даних організації, де інфраструктура бізнесу потребує прийняття рішень на місці, використовуючи хмарні обчислення.

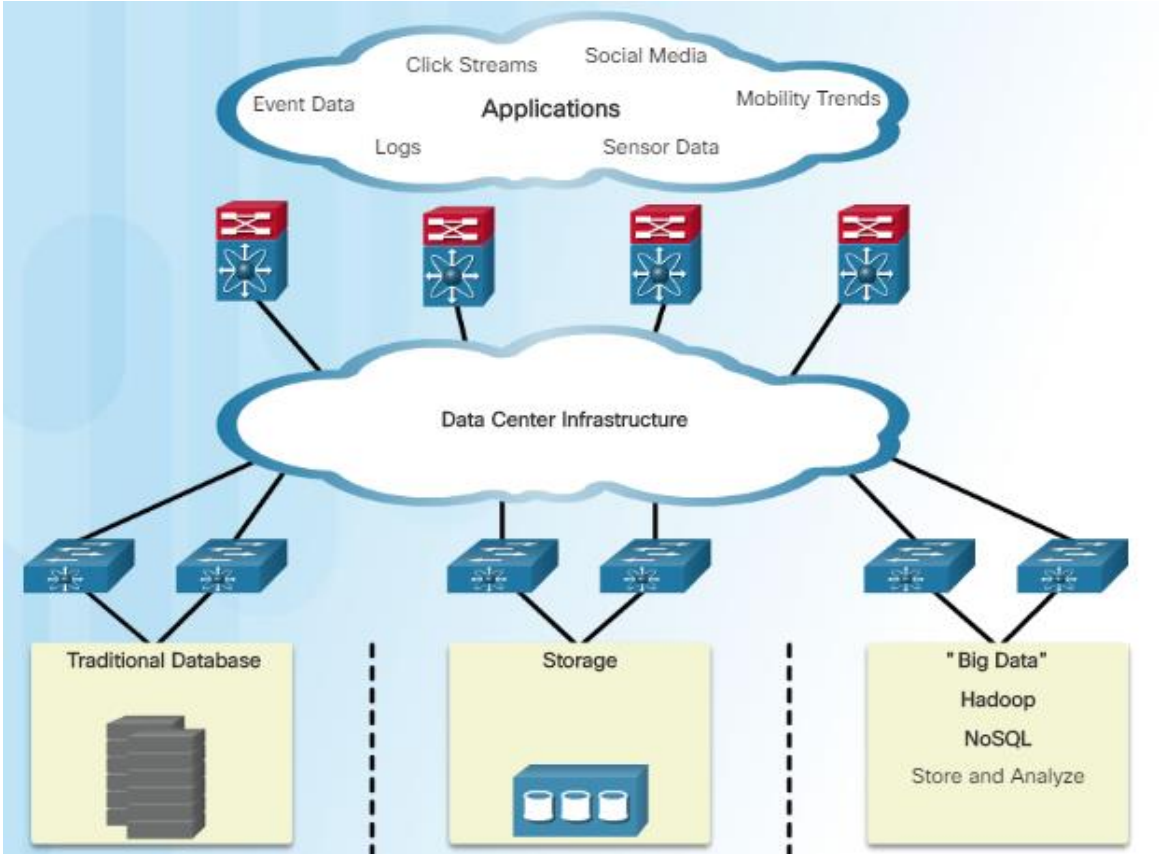


Рис. 1.8. Інфраструктура великих даних [2]

1.12. Розподілені дані та їх обробка

Наступне покоління управління даними з'явилося за допомогою системи управління реляційними базами даних (Relational database management system, RDBMS). Протягом 30 років це був стандартний підхід до управління даними. Реляційні бази даних фіксують зв’язки між різними наборами даних, створюючи більш корисну інформацію.

Більшість комерційних рішень RDBMS використовують SQL в якості мови запитів до сьогодні. Прикладом запиту SQL є: SELECT id, ім'я, ціна інвентаря, де ціна, наприклад, є <20. Приклади продуктів, які використовують структуровану мову запитів для доступу до даних, включають MySQL, SQLite, MS SQL, Oracle та IBM DB2.

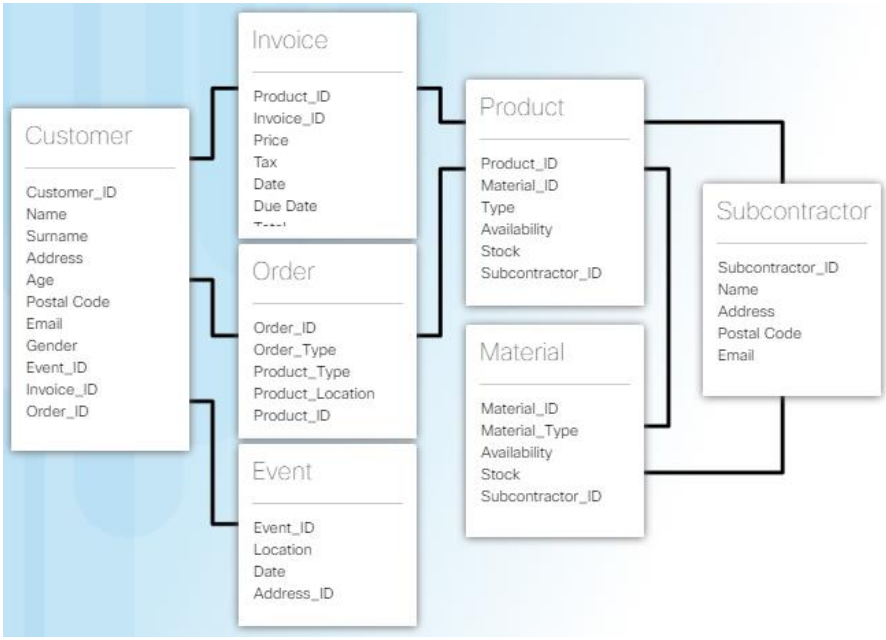


Рис. 1.9. Приклад реляційної бази даних [2]

Ще одна характеристика реляційних баз даних – це відмінність між базою даних та системою управління, що використовується для запиту бази даних. Зазвичай із RDMS та базовою базою даних багато користувачів можуть одночасно запитувати реляційну базу даних. Користувач зазвичай не знає всіх відносин, що існують усередині бази даних, адже, скоріше за все, користувач резюмує представлення бази даних, яка відповідає його потребам.

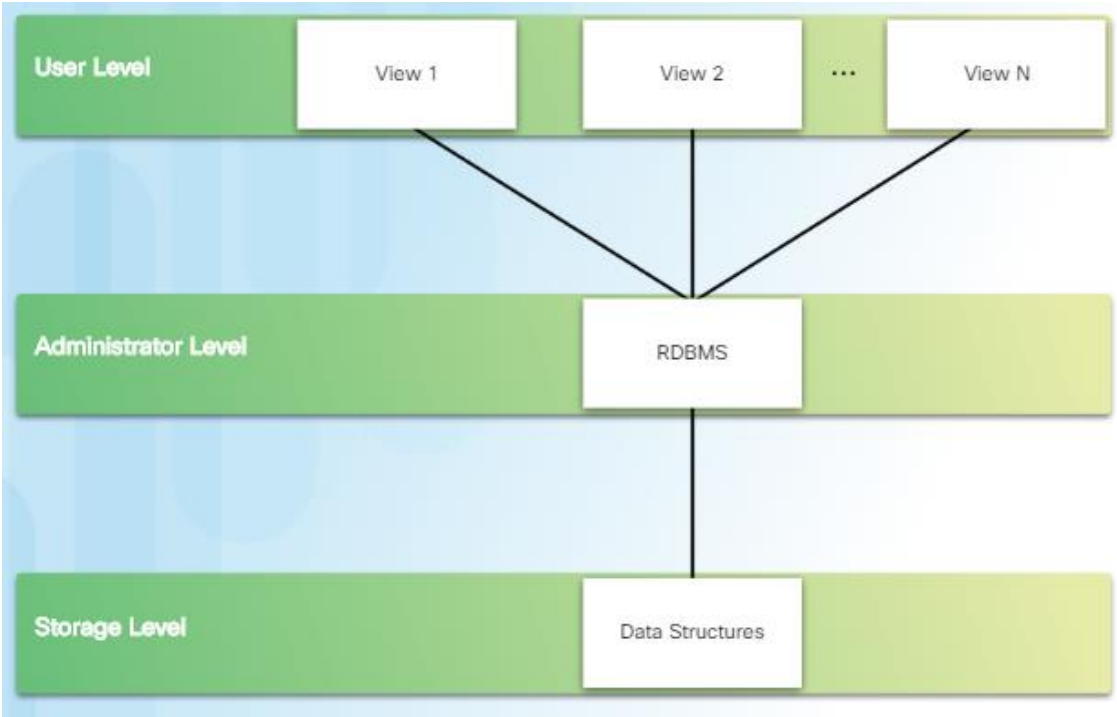


Рис. 1.10. Абстрагування даних у реляційній базі даних [2]

Найнижчий рівень абстрагування описує, як фізично зберігаються дані. Наступний рівень описує, які дані зберігаються та зв’язки між даними. Це рівень, на якому працює адміністратор бази даних. Користувацький рівень – це найвищий рівень, який описує, до якої частини бази даних певний користувач або група користувачів можуть отримати доступ. У будь-який момент часу може бути визначено багато різних одночасних підключень до бази даних. Нереляційні бази даних SQL (NoSQL) дуже добре масштабуються з розподіленими базами даних, оскільки NoSQL може обробляти великі дані та веб-додатки в режимі реального часу краще, ніж RDBMS, запити до баз даних NoSQL зосереджені збору документів, таких як інформація, зібрана з веб-сайтів. NoSQL також дозволяє кластерам машин обробляти дані та забезпечувати кращий контроль над їх наявністю. Бази даних NoSQL широко використовуються для вирішення бізнес-проблем.

З точки зору управління даними, аналітика була простою, коли дані створювали лише люди. Обсяг даних був керований. Реляційні бази даних обслуговують потреби аналітиків даних. Однак, з поширеністю систем автоматизації бізнесу та вибуховим зростанням веб-додатків та даних, що генеруються, аналітиці стає важче керувати лише рішенням RDBMS. Приблизно 90% даних, які існують сьогодні, були зібрані лише за останні два роки. Цей збільшений обсяг за короткий проміжок часу є властивістю експоненціального зростання. Цей великий обсяг даних важко обробити та проаналізувати. Замість того, щоб великі бази даних оброблялися великими та потужними комп'ютерами мейнфрейму та зберігалися в гігантських дискових масивах (вертикальне масштабування), розподілена обробка даних приймає великий об'єм даних і розбиває їх на менші частини. Ці менші обсяги даних поширюються у багатьох місцях, які обробляються багатьма комп'ютерами з меншими процесорами. Кожен комп'ютер у розподіленій архітектурі аналізує свою частину зображення Big Data (горизонтальне масштабування).

Більшість розподілених файлових систем розроблені таким чином, щоб вони не були помітні для клієнтських програм. Розподілена файлова система знаходить

файли та переміщує дані, але користувачі не можуть знати, що файли розподіляються між багатьма різними серверами чи вузлами. Користувачі отримують доступ до цих файлів так, ніби вони є локальними для власних комп'ютерів. Усі користувачі бачать однаковий вигляд файлової системи та отримують доступ до даних одночасно з іншими користувачами.

Hadoop був створений для вирішення цих великих обсягів даних. Проект Hadoop розпочався з двох граней: розподілена файлова система Hadoop (Hadoop Distributed File System, HDFS) – це розподілена файлова система і MapReduce, який є розподіленим способом обробки даних. Hadoop перетворився на всеосяжну екосистему програмного забезпечення для управління великими даними. Є багато інших програм з розподіленою файловою системою (DFS). Ось лише декілька з них: Ceph , GlusterFS та файлова система Google.

База даних NoSQL зберігає та отримує доступ до даних інакше, ніж реляційні бази даних. NoSQL іноді називають "не тільки SQL", "не SQL" або "нереляційними". Системи NoSQL можуть підтримувати SQL-подібні мови запитів. Бази даних NoSQL використовують структури даних, такі як ключ-значення, широкий стовпчик, графік або документ. Багато баз даних NoSQL надають "можливу послідовність". З можливою послідовністю зміни бази даних з часом з’являються у всіх вузлах. Це означає, що запити для даних можуть не надавати останню доступну інформацію. Причиною створення NoSQL було спрощення дизайну баз даних. Легше масштабувати кластери вузлів за допомогою NoSQL, ніж це виконувати у стандартних реляційних базах даних. Найпопулярнішими базами даних NoSQL у 2015 році були MongoDB, Apache Cassandra та Redis.

Структурована мова запитів (SQL) призначена для управління, пошуку та обробки даних, включаючи Big Data. SQLite – це бібліотека, яка використовує автономний, транзакційний двигун бази даних SQL. Код для SQLite знаходиться у відкритому доступі, що означає, що він може вільно використовуватись у комерційних та приватних цілях. SQLite – це найбільш широко розгорнута база даних у світі. SQLite також є вбудованою системою баз даних SQL. На відміну від

більшості інших баз даних SQL, у SQLite немає окремого серверного процесу. SQLite читає і записує безпосередньо у звичайні файли диска.

SQLite – популярний вибір для двигуна баз даних у мобільних телефонах, MP3-програвачах, приставках та інших електронних гаджетах. SQLite має невеликий слід коду, дозволяє ефективно використовувати пам’ять, дисковий простір та пропускну здатність диска, відрізняється високою надійністю і не потребує обслуговування адміністратора бази даних. SQLite часто використовується замість RDBMS для тестування. SQLite не потребує налаштувань, що значно спрощує тестування.

Розглянемо кілька корисних функцій SQLite.

- Ніяких налаштувань чи адміністрування не потрібно. Він має простий у користуванні API.
- Повна база даних зберігається в одному файлі міжплатформних дисків. Може використовуватися як формат файлу програми.
- Має невеликий слід коду.
- Це крос-платформна SQL. Підтримує Android, iOS, Linux, Mac, Windows та кілька інших операційних систем.
- Джерела для SQLite знаходяться у відкритому доступі.
- Має окремий інтерфейс командного рядка (CLI).
- Усі зміни в межах однієї транзакції відбуваються повністю або зовсім не відбуваються. Це справедливо навіть у випадку збою програми чи операційної системи або відмови живлення.

Використання технологій SQL та баз даних є ефективним для отримання підмножини даних із наявного набору даних, що зберігається в базі даних. Вираз SQL, який виконує цю дію, називається SQL-запитом. У бізнесі багато важливих проблем не вдається вирішити за допомогою простого запиту SQL і потрібен більш складний аналітичний процес. Ось тут використовується більш потужна мова програмування для аналізу даних, така як R або Python. R і Python мають великі спільноти розробників. Їх користувачі відомі тим, що розробляють модулі

аналізу даних та безкоштовно надають їх спільноті. Через це будь-який користувач може завантажувати та використовувати попередньо запрограмовані модулі та інструменти. Можливість створювати інструменти аналізу даних з нуля дозволяють отримати спеціально налаштовані програми. Процес створення інструменту аналізу даних з нуля можна розділити на дві основні частини: модель та код.

Моделювання полягає у визначенні того, що робити з даними для досягнення бажаних результатів та висновків. Припустимо, потрібно створити персональний фітнес-трекер та не існує попередньо запрограмованого модуля, який би виконував саме те, що ми хочемо зробити. Модуль трекера містить акселерометр, який є датчиком, здатним вимірювати прискорення пристрою. Акселерометр можна використовувати для визначення швидкості та напрямку руху. Швидкість і напрямок руху пристрою завжди відповідають швидкості та напрямку його користувача, коли він прикріплений до користувача.

Але що робити, якщо прилад кріпиться до ваги гантелей чи тенісної ракетки? Пристрій все одно буде отримувати однакові дані, швидкість і напрямок руху, але через різні програми інтерпретацію цих даних слід адаптувати до нового використання. У цьому контексті моделювання можна розглядати як спосіб інтерпретації та обробки даних. Наприклад, якщо фітнес-трекер прикріплений до користувача, дві поспіль точки без руху (швидкість дорівнює нулю), ймовірно, представляють початок і кінець спринту. Якщо вони прикріплені до ваги гантелей, ймовірно, дані представляють момент, коли гантель був піднятий з підлоги та найвищою точкою, яку користувач зміг підняти перед тим, як повернути його на підлогу.

Код є другою частиною створення інструментів аналізу даних з нуля. Код – це програма, яка обробляє дані і повинна бути записана відповідно до створеної моделі. Хоча модель і код є двома окремими об'єктами, вони пов'язані, оскільки код побудований на основі моделі.

Висновок до лекції 1

Даними можуть бути слова в книзі, вміст електронної таблиці, фотографії, файли або потоки вимірювань, надіслані пристроєм. Чотири Vs великих даних – це об'єм, швидкість, різноманітність та достовірність. Структуровані дані – це дані, введені у фіксовані поля файлу бази даних або записі. Неструктуровані дані не мають фіксованої схеми, яка визначає тип даних. Дані в стані спокою – це статичні дані, що зберігаються у фізичному місці. Дані в русі аналізують та витягують значення з даних до їх збереження. Hadoop був створений для роботи з великими обсягами даних. База даних NoSQL зберігає та отримує доступ до даних інакше, ніж реляційна база даних.

Питання для закріплення

1. Який вплив Інтернету Речей та зростання даних?
2. Для чого використовується платформа Kaggle?
3. Яке визначення великих даних?
4. Наведіть приклади великих даних у реальному світі.
5. Які дані є відкритими?
6. Які дані є структурованими та неструктурованими?
7. Що таке хмарні та туманні обчислення?
8. Опишіть дані в спокої та дані в русі.
9. Якою є інфраструктура великих даних?
10. Яка роль Hadoop в обробці розподілених даних?

Список рекомендованої літератури

1. Byte Size Infographic: Visualising data // Електронний ресурс. Режим доступу: https://www.redcentricplc.com/resources/infographics/byte-size/
2. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
3. Kaggle// Електронний ресурс. Режим доступу: https://www.kaggle.com/
4. DrivenData // Електронний ресурс. Режим доступу: https://www.drivendata.org/
5. Big Data: the 3 VS explained // Електронний ресурс. Режим доступу: https://bigdataldn.com/intelligence/big-data-the-3-vs-explained/
6. Computing // Електронний ресурс. Режим доступу: https://home.cern/science/computing
7. Open Knowledge Foundation // Електронний ресурс. Режим доступу: https://okfn.org
8. Gapminder // Електронний ресурс. Режим доступу: https://www.gapminder.org
9. Портал відкритих даних України // Електронний ресурс. Режим доступу: https://data.gov.ua

Лекція 2.

Розроблення програмного забезпечення для аналізу веб-сайтів, які надають відкриті дані за допомогою Python Pandas. Відкриті дані, їх формати та засоби оброблення

План лекції

2.1. Можливості інструментів аналізу даних.
2.2. Роль Python в аналізі даних.
2.3. Традиційна аналітика великих даних та аналітика нового покоління.
2.4. Життєвий цикл аналізу даних.
2.5. Відкриті дані, їх формати та засоби обробки.
2.6. Веб-скрепінг.
2.7. Витягування, перетворення та завантаження даних.

2.1. Можливості інструментів аналізу даних

Інструмент, який потрібно використовувати, залежить від типу аналізу, який потрібно виконати. Деякі інструменти призначені для обробки маніпуляцій та візуалізації великих наборів даних. Інші інструменти розроблені з можливостями математичного моделювання для прогнозування. Незалежно від того, які інструменти використовуються, вони повинні відповідати наступним вимогам.

- **Простота використання** – інструмент, який легко вивчити та використовувати, часто є більш ефективним, ніж складний у використанні інструмент. Крім того, простий у використанні інструмент вимагає меншої кількості навчання та підтримки.
- **Маніпулювання даними** – програмне забезпечення повинно дозволяти користувачам очищати та змінювати дані, щоб зробити їх більш корисними. Це призводить до того, що дані є більш надійними, оскільки аномалії можна виявити, відкоригувати або видалити.
- **Спільний доступ** – дослідники повинні використовувати однакові набори даних, щоб мати змогу ефективно співпрацювати та інтерпретувати дані однаково.

- **Інтерактивна візуалізація** – щоб повністю зрозуміти, як змінюються дані з часом, важливо візуалізувати тенденції. Основні діаграми та графіки не можуть повною мірою представляти розвиток інформації так, як може виглядати теплова карта або перегляд руху часу.

2.2. Роль Python в аналізі даних

Існують різноманітні програми, які використовуються для форматування даних, їх очищення, аналізу та візуалізації. Багато компаній та організацій звертаються до інструментів з відкритим кодом для обробки та узагальнення даних. Мова програмування Python стала загальноприйнятим інструментом для оброблення даних.

Мова Python була створена у 1991 році як легка у вивченні мова програмування з багатьма бібліотеками, які використовуються для маніпулювання даними, машинного навчання та візуалізації даних. Завдяки використанню цих бібліотек програмістам не доводиться вивчати декілька мов програмування або витрачати час, вивчаючи, як використовувати різні програми для виконання функцій цих бібліотек. Python – це гнучка мова, яка зростає та стає все більш інтегральною для наукових досліджень завдяки гнучкості та простоті її вивчення.

Розглянемо основні бібліотеки Python для аналізу даних.

- **NumPy** – ця бібліотека додає підтримку роботи з масивами та матрицями. Має багато вбудованих математичних функцій для використання в наборах даних.
- **Pandas** – ця бібліотека додає підтримку таблиць та часових рядів. Використовується для маніпулювання та очищення даних.
- **Matplotlib** – ця бібліотека додає підтримку візуалізації даних. Це бібліотека для простих та складних 3D та контурних графіків.

2.3. Традиційна аналітика великих даних та аналітика нового покоління

До епохи великих даних роль часу в аналітиці даних обмежувалася тим, скільки часу знадобиться для складання набору даних з різних джерел або скільки часу потрібно для запуску набору даних за допомогою певного обчислення. З Big Data час стає важливим і в інших напрямках, оскільки значна частина даних отримується завдяки створенню можливостей негайно вжити заходів.

Дані постійно генеруються датчиками, споживачами товарів і послуг, користувачами соціальних мереж, реактивними двигунами, фондовим ринком і майже всім, що підключено до мережі. Ці дані не просто зростають у кількості, вони змінюються в режимі реального часу, що також вимагає аналізу даних в режимі реального часу під час збору даних.

Під час обговорення Big Data та прийняття рішень бізнесу на основі аналітики може покращити рентабельність інвестицій для бізнесу як функцію часу. Рішення, керовані даними, мають такі переваги:

- збільшено час на дослідження та розробку товарів та послуг;
- підвищення ефективності та швидше виготовлення та вихід на ринок;
- більш ефективний маркетинг та реклама.

Раніше, коли більшість наборів даних були відносно невеликими та керованими, аналітики могли використовувати традиційні засоби, такі як Excel або статистичну програму, таку як SPSS, для аналізу значущої інформації з цих даних. Зазвичай набір даних містив історичні дані, і обробка цих даних не завжди залежала від часу. Дані, якщо вони не надто великі, можна було очистити, відфільтрувати, обробити, узагальнити та візуалізувати за допомогою діаграм, графіків та інформаційних панелей. Зі збільшенням обсягів, швидкості та різноманітності наборів даних складність зберігання, обробки та агрегації даних стає проблемою для традиційних аналітичних інструментів. Великі набори даних можуть поширюватися та оброблятися на декількох географічно розсіяних фізичних пристроях, а також у хмарі. Для цих великих наборів даних потрібні

великі інструменти даних, такі як Hadoop та Apache Spark, щоб забезпечити аналіз у режимі реального часу та прогнозування моделювання.

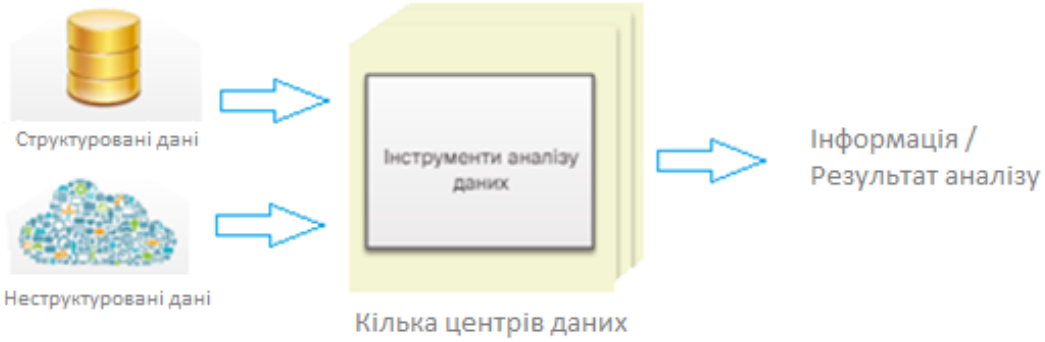


Рис. 2.1. Аналіз великих даних [1]

Для того, щоб підприємства приймали оптимальні рішення, вже недостатньо збирати дані попереднього фінансового року та виконувати описовий аналіз типів запитів. Все частіше необхідно використовувати інструменти прогнозного аналізу, щоб залишатися конкурентоспроможними у світі, у якому швидкість змін прискорюється. Аналітика наступного покоління не повинна покладатися виключно на здійснення статистичної аналітики для всього набору даних, як це робилося з традиційними інструментами. Через величезну кількість точок даних та атрибутів, новітні поведінки та уявлення можна отримати за допомогою вдосконаленого аналізу, що покращує точність прогнозування. Наприклад, можна відповісти на наступні запитання, щоб вносити корективи в реальний час:

• Які акції, швидше за все, матимуть найвищий щоденний приріст на основі торгів за останню годину?

• Який найкращий спосіб перевезти вантажні автомобілі на сьогодні в другій половині дня, виходячи з ранкових продажів, наявних запасів та поточних звітів про рух?

• Яке технічне обслуговування для літака базується на даних про продуктивність, отриманих під час останнього польоту?

Обробка даних, згенерованих машиною, у поєднанні з географічним розмахом дуже масштабних систем, кількістю пристроїв, що генерують дані, різноманітністю виробників пристроїв, частотою генерування даних та загальним

обсягом даних вимагає нового інфраструктурного програмного забезпечення. Це інфраструктурне програмне забезпечення повинне бути здатним розподіляти обчислення та зберігання даних між краєм, туманом та хмарою там, де це краще відповідає потребам бізнесу.

Рис. 2.2. Аналітика нового покоління [1]

Аналітика даних нового покоління дозволяє бізнесу краще розуміти вплив своїх продуктів і послуг, коригувати їх методи та цілі та швидше забезпечувати своїх клієнтів кращими продуктами. Можливість отримувати нові дані зі своїх даних приносить ділову цінність.

2.4. Життєвий цикл аналізу даних

Існує багато методологій проведення аналізу даних, включаючи популярний міжгалузевий стандартний процес обробки даних (CRISP-DM), який використовується більш ніж 40% аналітиків даних. Близько 27% аналітиків даних використовують власну методологію. Решта використовують інші методики. Максимально схожий на науковий метод, життєвий цикл аналізу даних розроблений для використання в бізнес-середовищі. Стрілки спрямовані в обидва боки між деякими кроками. Це підкреслює той факт, що життєвий цикл може зажадати багатьох ітерацій, перш ніж ті, хто приймає рішення, будуть досить впевнені, щоб рухатися вперед.

Рис. 2.3. Життєвий цикл аналізу даних [1]

Як і в науковому методі, життєвий цикл аналізу даних починається з питання. Наприклад, ми можемо задати питання: «Який злочин був найбільш поширеним у Києві, 20 серпня 2015 року?» Кожен крок життєвого циклу аналізу даних включає багато завдань, які необхідно виконати, перш ніж перейти до наступного кроку. Нижче наведено короткий опис кожного кроку.

- **Збір даних** – процес пошуку даних, визначення, чи є достатньо даних для завершення аналізу. Наприклад, ми шукаємо відкритий набір даних про злочинність для Києва протягом серпня 2015 року.
- **Підготовка даних** – цей крок може включати багато задач з перетворення даних у формат, відповідний інструменту, який буде використовуватися. Набір даних про злочини вже може бути підготовлений до аналізу. Однак зазвичай можна внести деякі коригування, які допоможуть відповісти на питання.
- **Вибір моделі** – цей крок включає вибір методики аналізу, яка найкраще відповість на питання із наявними даними. Після вибору моделі вибирається інструмент (або інструменти) для аналізу даних.
- **Аналіз даних** – процес тестування моделі на даних та визначення надійності моделі та аналізованих даних.

- **Представлення результатів** – це, як правило, останній крок для аналітиків даних. Це процес донесення результатів до осіб, які приймають рішення. Іноді аналітику даних просять рекомендувати дії. За даними про злочини 20 серпня, гістограма, кругова діаграма або якесь інше представлення можуть бути використані для повідомлення про те, який злочин найбільш поширений. Аналітик може запропонувати посилити присутність поліції у певних районах, щоб стримувати злочинність у конкретний день, наприклад, 20 серпня.
- **Прийняття рішень** – заключний крок у життєвому циклі аналізу даних. Організаційні лідери включають нові знання як частину загальної стратегії. Процес починається заново зі збору даних.

2.5. Відкриті дані, їх формати та засоби обробки

Є багато різних джерел даних. Велику кількість даних можна знайти у таких файлах, як документи MS Word, електронні листи, електронні таблиці, MS PowerPoints, PDF-файли, HTML та файли простого тексту. Це лише декілька типів файлів, які містять дані. Великі дані також можна знайти в державних та приватних архівах. Скановані паперові архіви, що містять історичні дані з різних джерел, безумовно, є Big Data. Наприклад, існує величезна кількість даних у формах медичного страхування та рахунках-фактурах, ділових виписках та взаємодії з клієнтами, а також у податкових документах. Цей список є лише невеликою частиною архівованих даних.

Внутрішні вихідні дані для організацій створюються за допомогою систем управління взаємовідносинами з клієнтами, систем управління навчанням, систем і записів людських ресурсів, інтрамереж та інших процесів.

Різні програми створюють файли в різних форматах, не обов'язково сумісних один з одним. З цієї причини потрібен універсальний формат файлу. Файли значень, розділених комами (comma-separated values, CSV) – це тип простого тексту, викладений у стандарті RFC 4180.

CSV-файли використовують коми для розділення стовпців таблиці даних, а символ нового рядка – для розділення рядків. Кожен рядок – це запис. Хоча вони

зазвичай використовуються для імпорту та експорту в традиційних базах даних та електронних таблицях, конкретного стандарту немає.

JSON і XML – це також типи файлів у простому тексті, які використовують стандартний спосіб подання записів даних. Ці формати файлів сумісні з широким колом застосувань. Перетворення даних у загальний формат є цінним способом поєднання даних із різних джерел.

Рис. 2.4. Формати даних [1]

Інтернет – хороше місце для пошуку великих даних. Там можна знайти зображення, відео, аудіо. Публічні веб-форуми також створюють дані. Соціальні медіа, такі як YouTube, Facebook, обмін миттєвими повідомленнями, RSS та Twitter, додаються до даних, знайдених в Інтернеті. Більшість цих даних є неструктурованими, що означає, що класифікувати їх в базу даних непросто без певного типу обробки.

2.6. Веб-скрепінг

Веб-сторінки створені для надання інформації людям, а не машинам. Засоби "веб-скрепінгу" автоматично "витягують" дані з HTML-сторінок. Це схоже на веб-сканер або павук пошукової системи, який досліджує Інтернет для вилучення даних та створення бази даних, щоб відповідати на пошукові запити. Програмне забезпечення для скрепінгу веб-сторінок може використовувати протокол передачі гіпертексту або веб-браузер для доступу до мережі. Веб-сканування – це автоматизований процес, в якому використовується бот або веб-сканер. Конкретні

дані збираються та копіюються з Інтернету в базу даних або електронну таблицю. Дані можуть бути проаналізовані.

Для здійснення веб- скрепінгу спочатку потрібно завантажити веб-сторінку, а потім витягти з неї потрібні дані. Веб-скрепери зазвичай виймають щось із сторінки, щоб використовувати його з іншою метою в іншому місці (для пошуку та копіювання імен, номерів телефонів та адрес). Цей процес відомий як контактий скрепінг.

Окрім роботи з контактами, веб-скрепінг використовується для інших видів пошуку даних, таких як списки нерухомості, дані про погоду, дослідження та порівняння цін. Багато великих постачальників веб-сервісів, таких як Facebook, надають стандартизовані інтерфейси для автоматичного збору даних за допомогою API.

Найпоширеніший підхід – використання інтерфейсів прикладних програм RESTful (API). API RESTful використовують HTTP як протокол зв'язку та структуру JSON для кодування даних. Інтернет-сайти, такі як Google та Twitter, збирають велику кількість статичних даних та часових рядів. Знання API для цих сайтів дозволяє аналітикам даних та інженерам отримати доступ до великої кількості даних, які постійно генеруються в Інтернеті.

2.7. Витягування, перетворення та завантаження даних

Бази даних містять дані, вилучені, перетворені та завантажені (ETL – extract, transform, load). ETL – це процес "очищення" необроблених даних, щоб вони могли бути поміщені в базу даних. Дані часто зберігаються в декількох базах даних і повинні бути об'єднані в один набір даних для аналізу.

Більшість баз даних містять дані, які належать організації та є приватними.

Після доступу до даних з різних джерел, дані потребують підготовки до аналізу. Фахівці в галузі Data Science вважають, що підготовка даних може зайняти від 50 до 90 % часу, необхідного для проведення аналізу.

Оскільки дані, які включатимуть набір даних для аналізу, можуть надходити з різноманітних джерел, вони не обов'язково сумісні при їх поєднанні. Інша

проблема полягає в тому, що дані, які можуть бути представлені у вигляді тексту, повинні бути перетворені в числовий тип, якщо вони будуть використані для статистичного аналізу. Типи даних є важливими, коли для роботи з даними використовуються такі мови, як Python або R.

Окрім різних типів даних, один тип даних може бути відформатований по-різному, залежно від його джерела. Наприклад, різні мови можуть використовувати різні символи для представлення одного і того ж слова. Наприклад, британська англійська може використовувати різні написання, ніж американська англійська.

Формати часу та даних представляють складнощі. Хоча час і дати дуже конкретні, вони представлені в найрізноманітніших форматах. Час і дата є важливими для аналізу спостережень за часовими рядами. Тому вони повинні бути перетворені у стандартний формат, щоб аналіз мав якесь значення. Наприклад, дати можуть бути відформатовані у році, який спочатку слідує за днем та місяцем у деяких країнах, тоді як інші країни можуть представляти дані із місяцем, який слідує за днем та роком.

Аналогічно, час може бути представлений у 12-годинному форматі з позначенням AM та PM, або може бути представлений у 24-годинному форматі. Обговорюючи дані, ми можемо думати про ієрархію структур. Наприклад, сховище даних – це місце, яке зберігає безліч різноманітних баз даних таким чином, що можна отримати доступ до баз даних за допомогою тієї самої системи. База даних – це сукупність таблиць даних, які пов'язані одна з одною або декількома способами. Таблиці даних складаються з полів, рядків та значень, схожих на стовпці, рядки та комірки в електронній таблиці. Кожна таблиця даних може розглядатися як файл, а база даних – як колекція файлів. Інші структури даних або об'єкти використовуються Python. Наприклад, Python використовує рядки, списки, словники, кортежі та набори як основні структури даних. Кожна структура даних має власну групу функцій або методів, які можна використовувати для роботи з об'єктом. Крім того, популярна бібліотека аналізу даних Python під назвою "pandas " використовує інші структури даних, такі як дата

фрейми даних. Значна частина даних, які будуть розміщені в базі даних, щоб потім їх можна було запитати, надходить з різних джерел і в широкому діапазоні форматів.

Витягування, перетворення та завантаження (ETL) – це процес збору даних із різноманіття джерел, перетворення даних, а потім завантаження даних у базу даних. Дані однієї компанії можна знайти в документах Word, електронних таблицях, простому тексті, PowerPoints, електронних листах та PDF-файлах. Ці дані можуть зберігатися на різних серверах, які використовують різні формати.

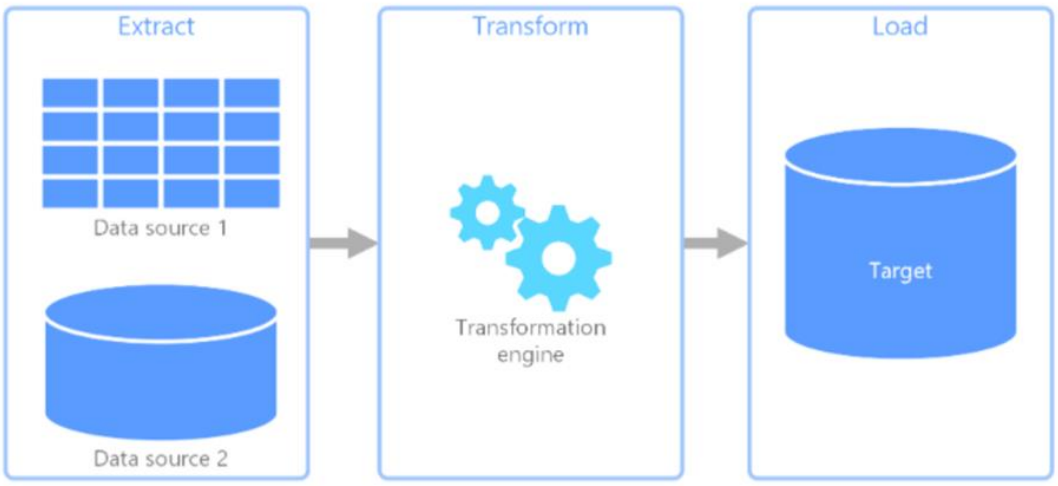


Рис. 2.5. Витягування, перетворення та завантаження даних [2]

Процес ETL містить наступні три основні кроки.

Крок 1. Витягування – дані збираються з кількох джерел.

Крок 2. Перетворення – після того, як дані будуть зібрані, вони повинні бути перетворені. Перетворення даних може включати агрегування, сортування, очищення та приєднання даних.

Крок 3. Завантаження – трансформовані дані завантажуються в базу даних для запитів.

Наведені вище описи трьох етапів процесу ETL спрощені. Насправді, перед тим, як дані можна завантажувати в базу даних, а потім запитувати їх, потрібно виконати дуже багато роботи. Етап вилучення збирає потрібні дані з джерела та робить їх доступними для обробки. Видобування перетворює дані в єдиний формат, який готовий до перетворення.

Наприклад, поєднання даних з сервера NOSQL та Oracle DB надасть дані в різних форматах. Ці дані повинні бути перетворені в єдиний формат. Також дані повинні бути перевірені, щоб переконатися, що вони мають бажаний тип інформації (значення). Це робиться за допомогою правил перевірки. Якщо дані не відповідають правилам перевірки, вони можуть бути відхилені. Іноді ці відхилені дані виправляються та підтверджуються.

Під час вилучення всі необхідні дані з джерела (джерел) отримують за допомогою мінімальних обчислювальних ресурсів, щоб не впливати на роботу мережі або комп'ютера.

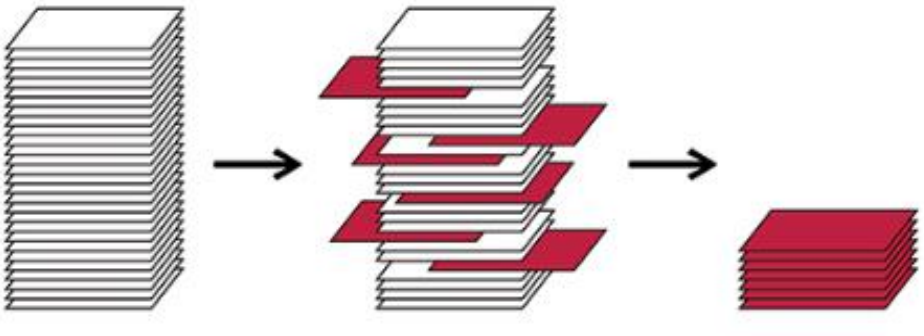


Рис. 2.6. Процес «витягування» даних [1]

На етапі перетворення використовуються правила для перетворення вихідних даних у тип даних, необхідних для цільової бази даних. Сюди входить перетворення будь-яких виміряних даних в один і той же вимір. Крок трансформації також вимагає декількох інших завдань. Деякі з цих завдань – це об'єднання даних із кількох джерел, їх агрегування, сортування, визначення нових значень, які обчислюються з агрегованих даних та застосування правил перевірки. Дані (включаючи деякі відхилені дані) можуть пройти через іншу частину етапу перетворення, відомий як «очищення» даних. Частина очищення етапу трансформації додатково забезпечує узгодженість вихідних даних. Крок завантаження – це коли трансформовані дані завантажуються в цільову базу даних.

Деякі організації можуть перезаписати наявні дані накопичувальними даними. Завантаження нових трансформованих даних може здійснюватися щогодини, щодня, щотижня або щомісяця. Це може статися лише тоді, коли в

трансформованих даних відбулася певна кількість змін. Під час кроку завантаження застосовуються правила, визначені у схемі бази даних. Деякі з цих правил перевіряють унікальність та послідовність даних, поля, які мають обов'язкове володіння, мають необхідні значення тощо. Ці правила допомагають забезпечити успішність завантаження та подальших запитів даних.

Висновок до лекції 2

Дані сьогодні не доцільно зберігати на кількох машинах та обробляти лише одним інструментом. Спеціалісти, що приймають рішення, все частіше покладаються на аналітику даних, щоб витягнути необхідну інформацію в потрібний час, у потрібному місці та прийняти правильне рішення.

Прогнозна аналітика передбачає результати та пропонує курси дій, які матимуть найбільшу користь для організації. Файли, дані з мережі Інтернет, датчики та бази даних – це приклади джерел даних. Витягування, перетворення та завантаження (ETL) – це процес збору даних з різних джерел, перетворення даних, а потім завантаження даних у базу даних.

Питання для закріплення

1. Яка роль Python в аналізі даних?
2. Опишіть традиційну аналітику великих даних та аналітику нового покоління.
3. Опишіть життєвий цикл аналізу даних.
4. Опишіть відкриті дані, їх формати та засоби обробки.
5. Що таке веб-скрепінг?
6. Поясніть процеси витягування, перетворення та завантаження даних.

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
2. Extract, transform, and load (ETL) // Електронний ресурс. Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl

Лекція 3.

Форматування даних про час та дату, читання та запис файлів в Python. Взаємодія із зовнішніми додатками

План лекції

3.1. Форматування даних про час та дату у Python.
3.2. Читання та запис файлів в Python.
3.3. Взаємодія із зовнішніми додатками.

3.1. Форматування даних про час та дату у Python

Модуль Python, який використовується для оброблення даних про час та дату, називається **datetime** [1]. Модуль **datetime** включений у більшість дистрибутивів Python як стандартна бібліотека; однак його потрібно імпортувати для використання у коді. Особливості модуля **datetime** представлені об'єктно-орієнтованою парадигмою програмування. Модуль складається з класів часу та дати. У кожного класу є свої методи, які можна викликати для роботи з екземплярами класів, які називаються об'єктами [2].

Рис.3.1 ілюструє застосування деяких основних об'єктів і методів, і включений в модуль **datetime**.

Datetime Module Terminolgy

Term	Example	Use
module	datetime	import **datetime** as dt
class	time date datetime, etc.	dt.**time** dt.**date** dt.**datetime**
object	Variables t, d, and dateAndTime	**t** = dt.time(12,31,00) **d** = dt.date(1972,12,31) **dateAndTime** = dt.dateandtime.now()
method	strftime() weekday() isoformat	t.**strftime**("%H:%M:%S") d.**weekday**() dateAndTime.**isoformat**()

Рис. 3.1. Застосування деяких основних об'єктів і методів **datetime** [3]

Метод стрипінгу використовує серію кодів форматування або директив як своїх параметрів (рис. 3.2).

Directive	Meaning
%a	Locale's abbreviated weekday name
%A	Locale's full weekday name
%b	Locale's abbreviated month name
%B	Locale's full month name
%c	Locale's appropriate date and time representation
%d	Day of the month as a number [01,31]
%H	Hour (24-hour clock) as a number [00,23]
%I	Hour (12-hour clock) as a number [01,12]
%j	Day of the year as a number [001,366]
%m	Month as a number [01,12]
%M	Minute as a number [00,59]
%p	Locale's equivalent of either AM or PM
%S	Second as a number [00,61]
%w	Weekday as a number [00,61]
%W	Week number of the year (Monday as the first day of the week) as a number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.
%y	Year without century as a number [00,99]
%Y	Year with century as a number
%Z	Time zone name (no characters if no time exists)

Рис.3.2. Список кодів форматування модуля **datetime** [3]

На рис. 3.3 показаний код Python, який використовує модуль **datetime** для подання дати та часу у форматі, який зазвичай використовується у США.

```
#load the datetime module as dt
import datetime as dt

#create a datetime object that contains the current time
currentDT = dt.datetime.now()

#view the value of currentDT
print(currentDT)
```

```
#create a new string object that contains the reformatted date and time
UDdt = currentDT.strftime('%b %d, %Y %I:%M %p')
#display the result
UDdt
```

Рис.3.3. Використання модуля **datetime** [3]

3.2. Читання та запис файлів в Python

Модуль **csv** є частиною стандартної бібліотеки Python. Модуль **csv** дозволяє читати та записувати у .csv файли. Python також має основні методи створення, відкриття та закриття зовнішніх файлів. Таблиці даних будуть існувати лише в оперативній пам'яті, поки вони не будуть збережені у файли.

Метод **open ()** використовується для створення нового файлу або для відкриття наявного файлу, який буде містити дані, які потрібно зберегти. Функція **close ()** видаляє дані з буферів і закінчує функцію запису файлу для вказаного файлу. Важливо закрити всі файли, в які не слід записувати дані. Це зберігає системні ресурси та захищає файл від пошкодження. На рис. 3.4 показаний синтаксис функції **open ()** та пояснено деякі важливі значення, які можна надати методу. Він також ілюструє використання методу **close ()**. Цей код створює файл, закриває його, а потім знову відкриває його в режимі додавання, щоб дані могли бути додані до файлу.

```
myFile = open("newText.txt", "w")
myFile.close()
myFile = open("newText.txt", "a")

print(myFile)
```

```
<open file 'newText.txt', mode 'a' at 0x729e2338>
```

Рис.3.4. Використання методів **open ()** та **close ()** модуля **datetime** [3]

Відкриття неіснуючого файлу в режимі "a" створить цей файл так само, як і в режимі "w". Різниця лише в тому, де знаходиться вказівник. Він буде вказувати на початок файлу або на кінець файлу.

На малюнку 3.5 пояснюються деякі важливі значення, які можна надати методу **open ()** . Ці параметри можуть бути об'єднані (символ "+"), щоб вказати, що повинні використовуватися як режими читання, запису та додавання.

```
open('myFile.txt', "w")
```

File open() Method Modes

Modes	Description
w	Opens an existing file that has the file name specified. If the file does not exist, it opens a new file with that name.
r	Opens an existing file in read only mode.
a	Opens an existing file and will append new data to the end of the file.

Рис.3.5. Параметри методу **open ()** [3]

Дані можна записати у файл, використовуючи метод **write ()**. Якщо файл було відкрито в режимі "a", дані будуть додані до кінця файлу. У формат, можливо, потрібно буде додати символи «\» для форматування.

Наприклад, \ n або \ r \ n символи додадуть розриви рядків у кінець записаного рядка даних. Метод файлу **read ()** читає вміст відкритого файлового об'єкта. Це показано на рис. 3.6.

```
myFile = open('newText.txt', "w")
myFile.close()

myFile = open('newText.txt', "a")
myFile.write('Line 1\n')
myFile.write('Line 2\n')
myFile.write('Line 3\n')

myFile.close()
myFile = open('newText.txt', "r")
myFile.read()
```

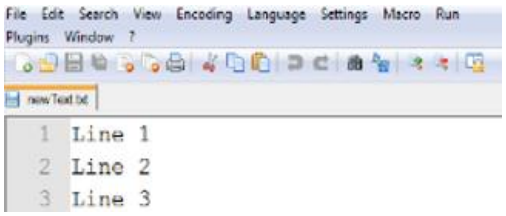


Рис. 3.6. Читання та запис даних у файл [3]

На рис. 3.6. у вхідній комірці один створюється новий файл. Файл закривається. У другій комірці файл знову відкривається в режимі додавання. У файл записуються три рядки тексту. У третій комірці файл закритий, щоб переконатися, що текст був записаний у файл. Файл знову відкривається в режимі читання, а метод **read ()** використовується для перегляду файлу. Файл також відображається у вигляді текстового редактора, який формує текст, використовуючи символи «\» для створення трьох окремих рядків.

3.3. Взаємодія із зовнішніми додатками

Python дозволяє взаємодіяти із зовнішніми додатками та операційною системою. Jupyter Notebook – це веб-програма з відкритим вихідним кодом, яка дозволяє створювати та обмінюватися документами, що містять діючий код, математичні рівняння, візуалізації та текст. Використання включає: очищення та

перетворення даних, чисельне моделювання, статистичне моделювання, візуалізація даних, машинне навчання та багато іншого [4].

У Jupyter Notebooks символ "!" дозволяє безпосередньо взаємодіяти з операційною системою. Наприклад, на рис. 3.7 показані дві команди Linux, виконані в Jupyter Notebooks. Команди починаються із символу "!".

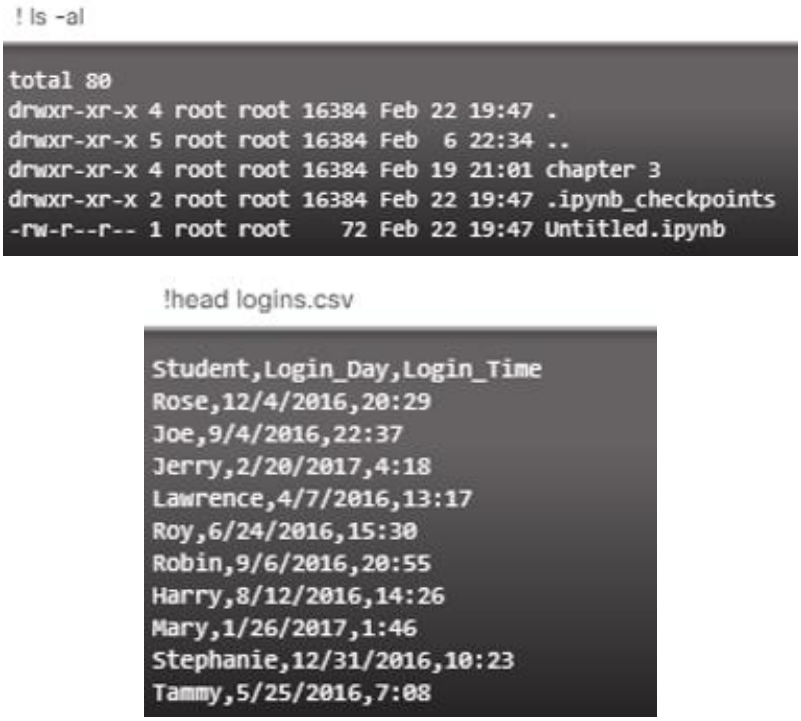


Рис. 3.7. Команди Linux, виконані в Jupyter Notebooks для взаємодії з операційною системою [3]

На рис. 3.8 показано використання модуля **subprocess** для зв'язку з зовнішньою програмою та збереження виводу команди, виданої цій програмі, в об'єкт Python. По-перше, створюється об'єкт для утримання команди, яку надсилає програма. У цьому випадку ми маємо намір надіслати команду до утиліти ping, яка доступна з оболонки Linux. Потім ми відправляємо цю команду, розділивши її на окремі слова, програмі методом **subprocess**. Нарешті, ми зберігаємо вихід команди в змінну і розділяємо її на рядок. Тоді ми можемо переглядати вміст об’єкта з друком та адресувати його окремі елементи за допомогою рядкової індексації.

```
'''import the subprocess library which is necessary for communication with external
apps'''
import subprocess
#We now execute the ping process as of from the shell:

pingCmd = 'ping -c 127.0.0.1'
process = subprocess.Popen(pingCmd.split(), stdout=subprocess.PIPE)
'''creat an object to hold the output of the process and split the output elements
into a list'''

process_output = process.communicate()[0]
process_output = process_output.split()
#view the contents of the output object
print(process_output)
```

```
['PING'. '127.0.0.1', '(127.0.0.1)', '56(84)', 'bytes', 'of', 'data.', '64', 'bytes',
 'from', '127..0.0.1', 'icmp_seq=1', 'ttl=64', 'time=0.094', 'ms', '64', 'bytes',
 'from', '127.0.0.1', icmp_seq=2', 'ttl=64', 'time=0.052', 'ms', '---', '127.0.0.1',
 'ping', 'statistics', '---', '2', 'packets', 'transmitted', '2', 'received,', '0%',
 'packet', 'loss,', 'time', '999ms', 'rtt', 'min/avg/max/mdev', '=',
 '0.052/0.073/0.094/0.021', 'ms']
```

```
#view the first five elements of the list
 process_output[0:5]
```

```
['PING', '127.0.0.1', '(127.0.0.1)', '56(84)', 'bytes']
```

Рис. 3.8. Використання модуля **subprocess** для зв'язку з зовнішньою програмою [3]

Висновок до лекції 3

Для оброблення даних про час та дату, використовується модуль Python **datetime**. Модуль **csv** дозволяє читати та записувати у .csv файли.

Python також має основні методи створення, відкриття та закриття зовнішніх файлів. Таблиці даних будуть існувати лише в оперативній пам'яті, поки вони не будуть збережені у файли. Метод **open ()** використовується для створення нового файлу або для відкриття наявного файлу, який буде містити дані, які потрібно зберегти. Функція **close ()** видаляє дані з буферів і закінчує функцію запису файлу для вказаного файлу. Python дозволяє взаємодіяти із зовнішніми додатками та операційною системою. Модуль **subprocess** використовується для зв'язку з зовнішньою програмою та збереження виводу команди, виданої цій програмі, в об'єкт Python.

Питання для закріплення

1. Як відбувається форматування даних про час та дату у Python?
2. Читання та запис файлів в Python?
3. Як відбувається взаємодія із зовнішніми додатками у Python?
4. Для чого використовується Jupyter Notebook?
5. Для чого використовується модуль *subprocess* у Python?

Список рекомендованої літератури

1. datetime – Basic date and time // Електронний ресурс. Режим доступу: https://docs.python.org/3/library/datetime.html
2. Python – Object Oriented // Електронний ресурс. Режим доступу: https://www.tutorialspoint.com/python/python_classes_objects.htm
3. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
4. Jupyter Notebook // Електронний ресурс. Режим доступу: https://jupyter.org/

Лекція 4.
Програмування Python та SQLite.
Призначення утиліти csvsql

План лекції

4.1. Основні операції SQL.
4.2. Робота Python з SQLite.
4.3. Призначення утиліти csvsql та метод execute().

4.1. Основні операції SQL

Існує багато способів роботи із зовнішніми файлами в Python. SQLite – це реалізація SQL, яка добре працює з Python. Замість того, щоб використовувати метод роботи з клієнтським сервером, використовуються з'єднання, встановлені між Python та базою даних SQL, створюючи об’єкт підключення SQL. Цей об’єкт матиме пов'язані з ним методи. Після створення об’єкта для з'єднання використовується метод створення об’єкта курсору. Об'єкт курсору має методи SQLite, доступні для виконання операцій SQL в базі даних.

SQL – мова для взаємодії з базами даних та таблицями. Існує ряд діалектів SQL, однак деякі основні операції є стандартними, вони працюють аналогічно, як у SQLite, MySQL чи інших реалізаціях SQL. SQLite може працювати в

інтерактивному режимі з командного рядка. Мова програмування Python, може взаємодіяти з SQLite через імпорт модулів.

SQL є мовою, що складається з трьох мов спеціального призначення. Перша – мова визначення даних (data definition language). Вона використовується для створення та маніпулювання структурою баз даних та таблиць SQL (табл. 4.1).

Табл. 4.1. Деякі загальні команди data definition language SQL

Команда	Пояснення
ALTER TABLE	Змінює структуру існуючої таблиці
CREATE DATABASE	Створює нову порожню базу даних
CREATE TABLE	Створює таблицю в рамках існуючої бази даних
DESCRIBE	Відображає структуру таблиці
DROP DATABASE	Повністю видаляє цілу базу даних
DROP TABLE	Видаляє таблицю з бази даних
USE	Відкриває базу даних, з якою слід працювати

Друга – мова маніпулювання даними (data manipulation language). Вона використовується для додавання, видалення або перетворення даних, які є в таблицях даних (табл. 4.2).

Табл. 4.2. Деякі загальні команди data manipulation language SQL

Команда	Пояснення
DELETE	Видаляє наявні дані
INSERT	Додає нові дані
REPLACE	Замінює записи, які мають дублікати даних, на записи, які потрібно вставити
UPDATE	Замінює значення у стовпцях даних новими значеннями залежно від зазначеного критерію

Третя є мовою запитів даних (data query language), яка використовується для доступу до даних у таблицях даних з метою генерування інформації (наприклад, команда SELECT отримує доступ до даних на основі заданого набору критеріїв).

4.2. Робота Python з SQLite

Розглянемо послідовність команд, які ілюструють основи операцій SQLite. По-перше, в операційну систему потрібно встановити зовнішній інструмент під назвою csvkit, щоб файл csv можна було імпортувати в базу даних SQLite. Нижченаведені команди відображають етапи створення бази даних SQLite, імпорту даних CSV в базу даних, виконання запиту в таблиці даних та перегляду результатів запиту.

```
import sqlite3 as sq1
```

Створюємо нову базу даних та встановлюємо з'єднання з нею:

```
conn = sq1.connect('logins.db')
```

```
!csvsql --db sqlite://///logins.db --insert logins.csv
```

Створюємо об'єкт cursor для запитів SQL:

```
cur = conn.cursor()
```

Створюємо запит SQL як рядковий об'єкт:

```
query = 'SELECT * FROM logins LIMIT 5'
```

Виконуємо запит SQL, об'єкт cursor тепер містить результат запиту:

```
cur.execute(query)
```

Створюємо цикл, який перебирає кількість рядків в об'єкті cursor та друкує їх вміст:

```
for row in cur:
    print(row)
```

```
(u'John', u'2016-12-29', u'1970-01-01 14:24:13.000000')
(u'Allan', u'2016-12-29', u'1970-01-01 03:16:54.000000')
(u'Robert', u'2016-12-30', u'1970-01-01 04:54:25.000000')
(u'Eve', u'2016-12-30', u'1970-01-01 08:32:14.000000')
(u'Leslie', u'2016-12-30', u'1970-01-01 20:34:54.000000')
```

Команда "!csvsql --db ..." може бути виконана як перша команда. Це зовнішній інструмент, який потрібно встановити в ОС. Для виконання цього командного рядка можна використовувати командний рядок (Linux CLI), але для спрощення речей цю зовнішню команду можна виконати безпосередньо з ноутбука шляхом префіксації команди "!" [1].

4.3. Призначення утиліти csvsql

Sqlcsv – простий інструмент командного рядка, який можна використовувати для вибірки дані з бази даних та експорту результат як CSV та вставки даних в базу даних із CSV. Працює лише з Python 3 [2]. У наведених нижче прикладах використовується наступна схема таблиці з MySQL:

```
CREATE TABLE testtable(
  id INT AUTO_INCREMENT PRIMARY KEY,
  int_col INT,
  float_col FLOAT,
  varchar_col VARCHAR(255) )
```

Розглянемо, як отримати доступ до бази даних із мови програмування Python. В інших мовах використовується майже однакова модель: імена бібліотек та функцій можуть відрізнятися, але концепції однакові. Ось коротка програма Python, яка вибирає широти та довготи з бази даних SQLite, що зберігається у файлі під назвою survey.db:

```
1.    import sqlite3
2.    connection = sqlite3.connect("survey.db")
3.    cursor = connection.cursor()
4.    cursor.execute("SELECT Site.lat, Site.long FROM Site;")
5.    results = cursor.fetchall()
6.    for r in results:
7.    print(r)
8.    cursor.close()
9.    connection.close()
```

Результат виконання програми:

```
(-49.85, -128.57)
(-47.15, -126.72)
(-48.87, -123.4)
```

Програма починається з імпорту sqlite3 бібліотеки. Якби ми підключалися до MySQL або іншої бази даних, ми імпортували б іншу бібліотеку, але всі вони надають однакові функції, так що решта нашої програми не повинна змінюватись (принаймні, не сильно), якщо ми перемикаємося з однієї бази даних на іншу. Рядок 2 встановлює підключення до бази даних. Оскільки ми використовуємо SQLite, все, що нам потрібно вказати, це ім'я файлу бази даних. Інші системи можуть вимагати від нас надання імені користувача та пароля. Потім рядок 3 використовує це з’єднання для створення об’єкту cursor. Подібно курсору в редакторі, його роль полягає у відстеженні того, де ми знаходимось у базі даних. У рядку 4 ми використовуємо цей курсор, щоб попросити базу даних виконати для нас запит. Запит пишеться в SQL і передається cursor.execute як рядок. Нам потрібно переконатися, що SQL правильно відформатований; якщо це не так, або якщо щось виконується не так, коли вона виконується, база даних повідомляє про помилку. База даних повертає результати запиту у відповідь на cursor.fetchall на рядок 5. Цей результат – це список із одним записом для кожного запису в наборі результатів; якщо ми прокрутимо цей список (рядок 6) і надрукуємо ці записи списку (рядок 7), ми побачимо, що кожен із них – кортеж з одним елементом для кожного поля, про яке ми просили. Нарешті, рядки 8 і 9 закривають наш курсор і наше з’єднання, оскільки база даних може одночасно тримати відкритою лише обмежену їх кількість.

Оскільки встановлення з'єднання вимагає часу, однак, ми не повинні відкривати з'єднання, виконувати одну операцію, потім закривати з'єднання, лише щоб знову відкрити його через кілька мікросекунд, щоб виконати іншу операцію. Натомість краще створити одне з’єднання, яке залишатиметься відкритим протягом усього терміну дії програми [3].

Висновок до лекції 4

SQLite – це реалізація SQL, яка добре працює з Python. Замість того, щоб використовувати метод роботи з клієнтським сервером, використовуються з'єднання, встановлені між Python та базою даних SQL, створюючи об’єкт підключення SQL. Після створення об’єкта з'єднання використовується метод створення об’єкта курсору. Об'єкт курсору має методи SQLite, доступні для виконання операцій SQL в базі даних. SQLite може працювати в інтерактивному режимі з командного рядка, мова Python також може взаємодіяти з SQLite через імпорт модулів.

Питання для закріплення

1. З яких трьох мов спеціального призначення складається SQL?
2. Наведіть основні команди мови SQL.
3. Яке призначення утиліти csvsql?
4. Як отримати доступ до баз даних із програм, написаних на Python?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
2. sqlcsv // Електронний ресурс. Режим доступу: https://pypi.org/project/sqlcsv/
3. Programming with Databases – Python // Електронний ресурс. Режим доступу: https://swcarpentry.github.io/sql-novice-survey/10-prog/index.html

Лекція 5.
Процедура імпорту даних із файлів у Pandas.
Імпорт даних з мережі Інтернет.
Засоби для кореляційного аналізу в Pandas

План лекції

5.1. Статистичні підходи до аналітики великих даних.
5.2. Використання Pandas.
5.3. Імпорт даних з файлів.
5.4. Імпорт даних з мережі Інтернет.
5.5. Описова статистика в Pandas.
5.6. Засоби для кореляційного аналізу в Pandas.

5.1. Статистичні підходи до аналітики великих даних

У аналітиці великих даних використовуються різні статистичні підходи. Як відомо, описова статистика описує вибірку. Це корисно для розуміння вибіркових даних та для визначення їх якості. При роботі з великою кількістю даних, що надходять із багатьох джерел, може виникнути багато проблем. Іноді точки даних можуть бути пошкоджені, неповні або повністю відсутні. Описова статистика може допомогти визначити, яка частина даних у вибірці є корисною для аналізу та визначити критерії для вилучення даних, які є невідповідними або проблемними. Графіки описової статистики є корисним способом швидкого судження про вибірку. Наприклад, для аналізу може бути обраний зразок твітів. Деякі твіти у зразку містять лише символи, а інші твіти містять символи та зображення. Ми можемо аналізувати твіти, які містять зображення або твіти без зображень. Це дозволить визначити недійсні твіти на основі дуже простого критерію. Точки даних, які не відповідають основним критеріям, будуть вилучені з вибірки до того, як розпочнеться аналіз.

Методи машинного аналізу дуже часто використовуються в аналітиці великих даних:

- **Кластерний аналіз** – використовується для пошуку груп спостережень, схожих між собою.
- **Асоціація** – використовується для пошуку спільних зустрічей значень для різних змінних.
- **Регресія** – використовується для кількісної оцінки взаємозв'язку між варіаціями однієї або декількох змінних, якщо такі є.

У машинному навчанні комп'ютерне програмне забезпечення або надається, або отримує власний набір правил, які використовуються для проведення аналізу. Методи машинного навчання можуть вимагати великої потужності обробки і стали життєздатними лише при наявності паралельної обробки.

На рис. 5.1. показана таблиця, що складається з двох полів. Одне поле містить змінну, а інше складається із статистики, яка описує значення цієї змінної. У

цьому прикладі десять учнів взяли вікторину на десять балів. Коли викладач аналізує бали, створюється розподіл балів, як показано у другій таблиці. Це виражає кількість разів, коли бал відбувся в класі. Ймовірність балу виражається у співвідношенні частоти балів до загальної кількості балів.

Студент	**Вікторина (10 балів)**	**Оцінка**	**Частота оцінки**	**Ймовірність балу**
Студент 1	6	1	0	0
Студент 2	7	2	0	0
Студент 3	7	3	0	0
Студент 4	8	4	0	0
Студент 5	7	5	1	0.1
Студент 6	9	6	1	0.1
Студент 7	10	7	4	0.4
Студент 8	8	8	2	0.2
Студент 9	7	9	1	0.1
Студент 10	5	10	1	0.1

Рис. 5.1. Приклад таблиць з оцінками учнів за виконану вікторину [1]

Розподіл частот складається з усіх унікальних значень змінної та кількості разів, коли значення наступають у наборі даних. У розподілах ймовірностей замість частот використовується пропорція часу, коли значення виникає в даних. Гістограма може представляти розподіл набору даних. У випадку дискретної змінної кожному біну гістограми присвоюється певне значення. У випадку безперервної дії кожен контейнер пов'язаний з діапазоном значень. В обох випадках висота бункера представляє кількість разів, коли значення змінної приймає задане значення або потрапляє в діапазон відповідно.

Представлення гістограми розподілу даних може приймати будь-яку форму. У разі безперервної змінної форма також залежатиме від ширини бункерів, тобто від їх діапазону. Деякі фігури можна моделювати за допомогою чітко визначених функцій, які називаються функціями розподілу ймовірності.

Функції розподілу ймовірностей дозволяють представляти форму всього розподілу набору даних, використовуючи лише невеликий набір параметрів, таких як середнє значення та дисперсія.

Функція розподілу ймовірностей, яка особливо підходить для відображення багатьох подій, що відбуваються в природі, – це Гауссовий, або нормальний розподіл, яке є симетричним і має форму дзвоника. Інші розподіли не симетричні. Вершина графіка може бути ліворуч або праворуч від центру. Ця властивість розподілу називається перекосом. Деякі розподіли матимуть два піки і відомі як бімодальні. Правий і лівий кінці графіка розподілу відомі як хвости. Однією з характеристик розподілів, яка дуже часто використовується, є міри центральної тенденції. Ці міри виражають значення, які має змінна, яка є найближчою до центральної позиції в розподілі даних. Загальні міри центральності – це середнє, медіана та мода. Мода вибірки даних – це значення, яке зустрічається найчастіше (рис. 5.2). Значення, які ближчі до центру розподілу, зустрічаються з більшою частотою.

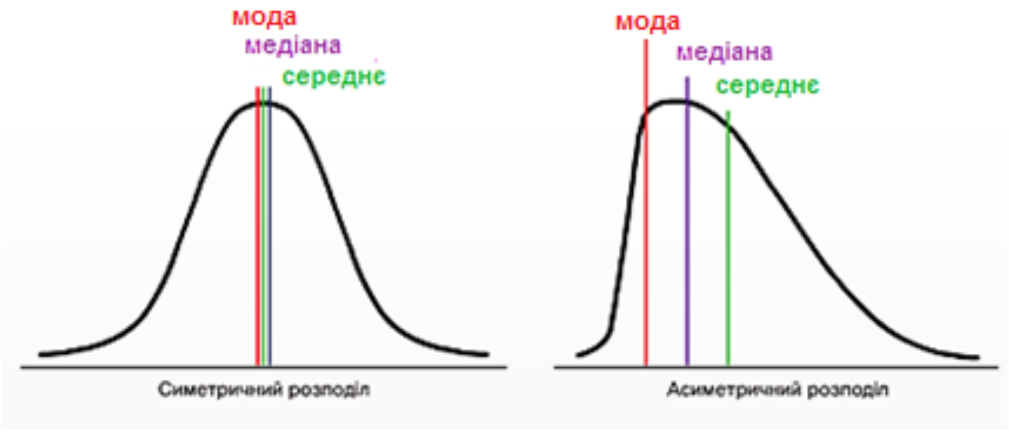


Рис. 5.2. Загальні міри центральності – це середнє, медіана та мода [1]

Середнє значення, також відоме як середнє, є найбільш відомим показником центральної тенденції. Дорівнює сумі всіх значень даних, поділених на кількість значень у наборі даних. Хоча середнє дуже часто використовується у повсякденному житті, вона, як правило, не є найкращим показником найбільш репрезентативного значення для розподілу. Наприклад, якщо в розподілі є незвично високі або низькі значення, на середнє можуть сильно впливають ті

крайні значення, які називаються викидами. Залежно від кількості випадаючих в наборі даних, середня величина або середня величина "перекошується" або змінюється в ту чи іншу сторону.

Медіана – це середнє значення в наборі даних після впорядкування списку значень (сортування). Медіана не чутлива до цих крайніх значень. Оскільки загальна кількість значень та фактичні значення у наборі даних однакові, середина у списку чи медіана залишається однаковою. Залежно від кількості випадаючих в наборі даних, середня величина або середня величина "перекошується" або змінюється в ту чи іншу сторону.

У той час як середнє значення час використовується для опису багатьох розподілів, воно залишає важливу частину загальної картини, яка є мінливістю розподілу. Наприклад, ми знаємо, що зовнішні значення можуть спотворювати середнє значення. Медіана наближає нас до того, що є головним у розподілі, проте ми все ще не знаємо, наскільки розподілені значення у вибірці. Основний спосіб опису змінності у вибірці – це обчислення різниці між найвищим і найнижчим значенням для змінної. Ця статистика відома як діапазон.

Дисперсія розподілу – це міра того, наскільки кожне значення в наборі даних від середнього. З дисперсією пов'язане стандартне відхилення, яке використовується для стандартизації розподілів як частини нормальної кривої, як показано на рис.5.3.

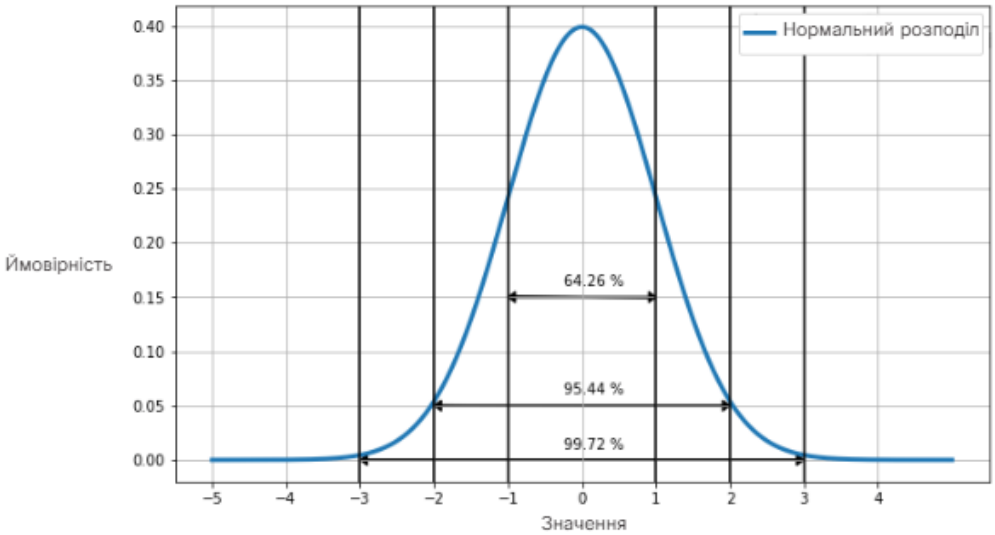


Рис. 5.3. Стандартні відхилення вибірки [1]

5.2. Використання Pandas

Pandas – це бібліотека з відкритим кодом для Python, яка додає високопродуктивні структури даних та інструменти для аналізу великих наборів даних. Структури даних Pandas включають структури рядів (series) та структури дата фреймів (dataframe). Дата фрейми є первинною структурою Pandas і є найбільш часто використовуваними. Дата фрейм схожий на електронну таблицю з рядками та стовпцями. Крім того, фрейми даних можуть мати додаткові індекси та стовпці, які є мітками для рядків та стовпців [2].

Дата фрейми легко будуються з ряду інших структур даних та зовнішніх файлів, таких як csv. Об'єктам кадрів даних доступний широкий спектр методів. Рядки та стовпці можуть маніпулювати різними способами, і оператори доступні для виконання математичних, рядкових та логічних перетворень на вміст фрейму даних (рис. 5.4).

	First Name	Last Name	Phone Number
1	Mary	Pratt	410-555-9697
2	Oscar	Milde	555-887-9547
3	Timmy	Thomas	471-555-9687

- two-dimensional
- labeled
- different types accomodated

Рис. 5.4. Компоненти дата фрейму Pandas [1]

Pandas імпортується в програму Python, використовуючи **import**, як і інші модулі. Зазвичай для використання імпорту **pandas as pd** для посилання на компоненти pandas простіше набрати. Нижче наведено код, необхідний для створення дата фрейму, який зображений на рис. 5.5.

```
import pandas as pd

data = {'First Name': ['Mary', 'Oscar', 'Timmy'],
        'Last Name': ['Pratt', 'Milde','Thomas'],
        'Phone Number': ['410-555-9697', '555-887-9547', '471-555-9687']}
directory = pd.DataFrame(data, columns=['First Name', 'Last Name', 'Phone Number'])
```

Directory

	First Name	Last Name	Phone Number
0	Mary	Pratt	410-555-9697
1	Oscar	Milde	555-887-9547
2	Timmy	Thomas	471-555-9687

Рис. 5.5. Створення дата фрейму вручну [1]

5.3. Імпорт даних з файлів

Великі набори даних збираються з різних джерел і можуть існувати у вигляді файлів різного виду. Створення дата фрейму Pandas шляхом кодування значень даних окремо не дуже корисно для аналізу великих даних.

Pandas включає деякі дуже прості у використанні функції для імпорту даних із зовнішніх файлів, таких як csv, у дата фрейми. Ми відтворимо дата фрейми телефонного каталогу, цього разу з більшого файлу csv. Pandas включає в себе функцію дата фрейму, який називається **read_csv ().**

Процедура така:

Крок 1. Імпортуйте модуль Pandas.

```
import pandas as pd
```

Крок 2. Перевірте, чи файл доступний у поточній робочій директорії. У цьому випадку команда **head** Linux використовується для перевірки файлу та попереднього перегляду його вмісту.

```
!head -n 5 directory.csv
```

```
First Name,Last Name,Phone Number
Mary,Pratt,410-555-9697
Oscar,Milde,555-887-9547
Timmy,Thomas,471-555-9687
John,Smith,252-959-8421
```

Крок 3. Щоб імпортувати файл в об'єкт дата фрейм, використовуйте метод Pandas **read_csv ().**

```
df_directory = pd.read_csv('directory.csv')
```

Крок 4. Використовуйте метод дата фрейму pandas **info ()**, щоб переглянути короткий зміст файлу.

```
df_directory.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15 entries, 0 to 14
Data columns (total 3 columns):
First Name      15 non-null object
Last Name       15 non-null object
Phone Number    15 non-null object
dtypes: object(3)
```

Крок 5. Відображення дата фрейму. Метод **head ()** використовується для відображення заголовків, індексу та значень для перших п'яти рядків.

```
df_directory.head()
```

Dataframe records

	First Name	Last Name	Phone Number
0	Mary	Pratt	410-555-9697
1	Oscar	Milde	555-887-9547
2	Timmy	Thomas	471-555-9687

5.4. Імпорт даних з мережі Інтернет

Імпортувати дані з Інтернету за допомогою Pandas дуже просто. Хоча для доступу до веб-даних доступно багато інтерфейсів прикладних програм (API), включаючи потокову передачу даних, статичні набори даних також можна отримати з Інтернету на основі URL-адреси файлу.

У прикладі, показаному на рис. 5.6, набір даних імпортується в набір даних із великої колекції Гуманітарної біржі даних [3]. Цей веб-сайт є гарним ресурсом для людей, зацікавлених у вивченні даних, пов'язаних з міжнародними гуманітарними проблемами.

```
import pandas as pd

url =
'http://manage.humdata.org/hdx/api/exporter/indicator/csv/TT014/source/mdgs/fromYear
/1950/toYear/0/language/en/TT014_Baseline.csv'

from_url = pd.read_table(url,sep=',')

from_url.head()
```

Country Data from CSV

	Country Code	Country Name	2015	2014	2013	2012	2011	2010	2009	2008	...	19
0	AFG	AFGHANISTAN	27.7	27.7	27.7	27.7	27.7	27.3	27.7	27.7	...	Na
1	AGO	ANGOLA	36.8	36.8	34.1	38.2	38.6	38.6	37.3	15.0	...	15
2	ALB	ALBANIA	20.7	20.0	15.7	15.7	16.4	16.4	7.1	7.1		Na

5 rows x 28 columns

```
from_url.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 192 entries, 0 to 191
Data columns (total 28 columns):
Country code    192 non-null object
Country name    192 non-null object
```

Рис. 5.6. Імпорт даних з мережі Інтернет у Pandas [1]

Ми імпортуємо набір даних, що містять інформацію про відсоток жінок, які працюють у національних парламентах для ряду країн протягом певних років. Процес включає в себе наступні кроки:

Крок 1. Імпорт Pandas.

Крок 2. Створення об'єкта рядка, який містить URL-адресу файлу.

Крок 3. Імпорт файлу в об'єкт дата фрейм методом pandas **read_table ().**

Крок 4. Перевірте імпорт за допомогою **head ()** та **info ()**. Висновок **info ()** вказує на кількість відсутніх значень (нульових записів), що є різницею між загальною кількістю записів та кількістю ненульових записів за кожен рік.

В мережі Інтернет є багато джерел даних. Наприклад, такі сайти, як Google і Twitter, мають API, які дозволяють підключати програми Python до потокових даних.

5.5. Описова статистика в Pandas

Pandas забезпечує дуже простий спосіб перегляду основних описових статистичних даних для фрейму даних. Метод **describe()** для об'єктів дата фрейму відображає наступне для числових типів даних:

- **count** – кількість значень, включених до статистики;
- **mean** – середнє значення;
- **std** – стандартне відхилення розподілу;
- **min** – найменше значення в розподілі;
- **25%** – значення для першого квартилю (25% значень знаходяться на рівні цього значення або нижче);
- **50%** – значення для другого квартилю або медіани (50% значень знаходяться на рівні цього значення або нижче);
- **75%** – значення для другого квартилю (75% значень знаходяться на рівні цього значення або нижче);
- **max** – найвище значення в розподілі.

5.6. Засоби для кореляційного аналізу в Pandas

Причинно-наслідкові зв’язки та кореляція – це типи зв’язків між умовами чи подіями. Причинно-наслідковий зв’язок – це відносини, у яких одна річ змінюється або створюється безпосередньо через щось інше.

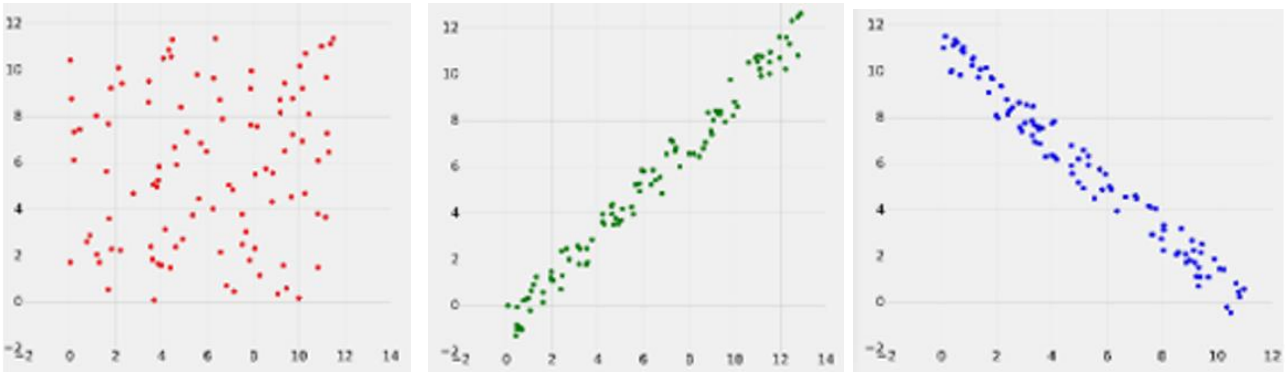
Наприклад, підвищення глобальної температури викликає зниження арктичної крижаної шапки. Це інтуїтивне відношення до явищ. Підвищення глобальної температури також може призвести до зменшення споживання вовни для використання у створенні теплого одягу. Чим тепліший клімат, тим менше буде попит на теплий одяг.

Кореляція – це зв’язок між величинами, у яких дві або більше величин змінюються з однаковою швидкістю. Наприклад, зменшуються глобальна температура та споживання вовни, ці величини змінюються з однаковою швидкістю та в аналогічному напрямку (обидві зменшуються).

Кореляції можуть бути позитивними та негативними. Позитивно корельовані величини змінюються в одному напрямку. Якщо одна величина збільшується, інша теж збільшується. Негативна кореляція виникає, коли величини змінюються в деякій схожій пропорції, але в протилежних напрямках. Іншими словами, якщо одна збільшується, інша зменшується аналогічно. Кореляції між величинами можна кількісно визначити, використовуючи статистичні підходи.

Найпоширенішою статистикою для обчислення кореляції є коефіцієнт кореляції Пірсона, це величина, яка виражається як значення між -1 і 1. Позитивні значення виражають позитивну залежність між змінами двох величин. Негативні значення виражають зворотну залежність.

Величина або позитивних, або негативних значень вказує на ступінь кореляції. Чим ближче значення до 1 або -1, тим міцніше зв'язок, 0 вказує на відсутність кореляції (рис. 5.7).



а) б) в)

Рис. 5.7 Розкиди невеликих наборів даних, які мають низьку, позитивну та негативну кореляцію, а) низька кореляція, r=0,0114, б) сильна позитивна кореляція, r=0,99, в) сильна негативна кореляція, r=-0,985 [1]

Метод **corr ()** простий у використанні. Розглянемо невеликий набір даних із більшого набору даних, який описує демографічні показники чисельності населення ряду міст.

```
import pandas as pd
csvName = 'povt-unemp.csv'
newFrame = pd.read_csv(csvName)
newFrame.head()
```

Poverty and Unemployment

	Poverty	%unempl
0	0.19	0.27
1	0.24	0.27

Дані спрощено, щоб містити лише два поля: відсоток людей, які живуть нижче межі бідності грошового доходу, і відсоток людей, які не працюють. Не дивно, що ці два поля повинні демонструвати сильну кореляцію. Це означало б, що ми очікували б, що для міста з великою кількістю людей, які живуть у злиднях, рівень безробіття також буде високим. У прикладі файл CSV, що містить дані, імпортується у дата фрейм. Дані швидко перевіряються за допомогою методів **head ()** та **describe()**, щоб переконатись, що імпорт працює належним чином. Для дата фрейму викликається метод **corr ()**. Результат відображається у таблиці кореляції. Бачимо, що з коефіцієнтом кореляції 0,73 безробіття має сильний зв’язок із бідністю.

```
newFrame.describe()
```

describe() Method Results

	Poverty	%unempl
count	81.000000	81.000000
mean	0.292099	0.362222

```
correl = newFrame.corr()

correl
```

corr() Method Results

	Poverty	%unempl
Poverty	1.000000	0.729233

Кореляції можна обчислити для кількох змінних одночасно. Це призведе до обчислення коефіцієнтів кореляції між усіма полями, що подаються в дата фрейм. Результатом може бути велика таблиця коефіцієнтів кореляції. Візуалізація, що називається **тепловою картою (heat map)**, корисна для розуміння того, як значення коефіцієнтів кореляції співвідносяться між собою. Графіки розсіювання корисні для швидкої візуалізації можливої кореляції у наборі даних. На тепловій карті поля в даних утворюють горизонтальну та вертикальну мітки для сітки значень (рис. 5.8).

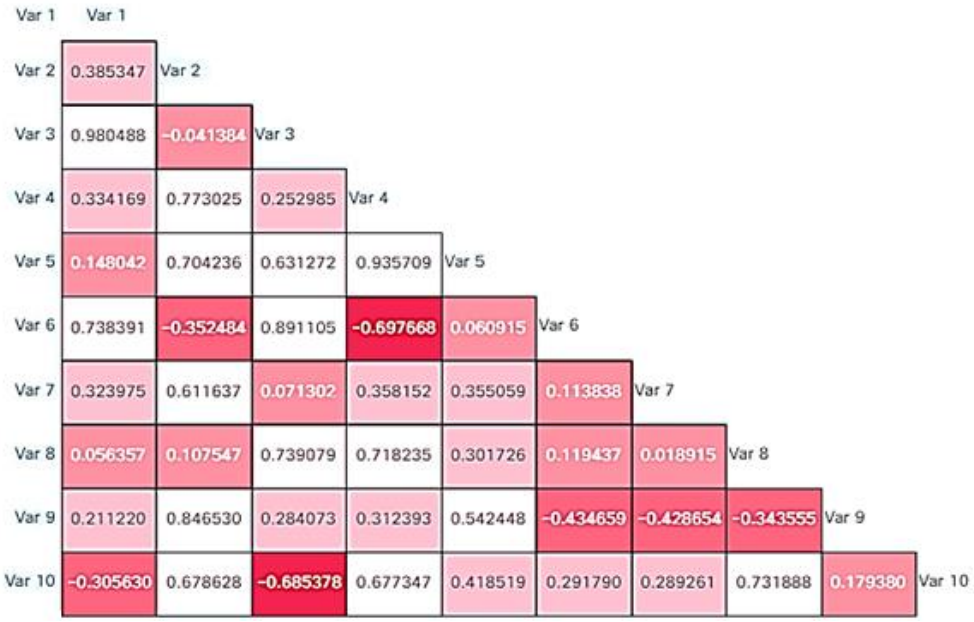


Рис. 5.8. Теплова карта коефіцієнтів кореляції між 10 змінними [1]

Кожне значення комірки є коефіцієнтом кореляції поля на горизонтальній розмірності сітки з полем на вертикальній осі.

Значення на перетині обраних розмірів є коефіцієнтом для цієї пари значень. Для інтерпретації даних значення кореляції мають кольорове кодування. Інтенсивність або відтінок кольору для кожного значення пропорційні цьому значенню. Наприклад, усі негативні коефіцієнти кореляції можуть бути представлені в червоному відтінку, а усі позитивні – у відтінку синього. Чим глибший колір, тим ближче значення до 1 або -1. Це допомагає зрозуміти значення з даних кореляції.

Висновок до лекції 5

Дослідницький аналіз даних дає описові та графічні узагальнення даних з для того, щоб з результатів виявити цікаві закономірності. Спостереження, змінні та значення мають вирішальне значення для аналізу.

Pandas – це бібліотека Python з відкритим кодом з інструментами для аналізу великих наборів даних, імпорту даних із файлів та з мережі Інтернет, перегляду описової статистики, пошуку статистичних залежностей для наборів даних. Дані зазвичай потребують очищення, перетворення та оброблення перед аналізом даних.

Питання для закріплення

1. Які статистичні підходи до аналітики великих даних ви знаєте?
2. Яке призначення бібліотеки Pandas?
3. Як відбувається імпорт даних з файлів у Pandas?
4. Як відбувається імпорт даних з мережі Інтернет?
5. Як перевірити описову статистику даних у Pandas?
6. Які засоби для кореляційного аналізу в Pandas ви знаєте?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

2. Pandas // Електронний ресурс. Режим доступу: https://github.com/pandas-dev/pandas

3. The Humanitarian Data Exchange // Електронний ресурс. Режим доступу: https://data.humdata.org/

4. Pandas // Електронний ресурс. Режим доступу: https://www.w3schools.com /python/pandas/pandas_intro.asp

Лекція 6.

Оброблення відсутніх даних. Перетворення типів даних та маніпулювання дата фреймами у Python

План лекції

6.1. Оброблення відсутніх даних.
6.2. Перетворення типів даних.
6.3. Маніпулювання дата фреймами.

6.1. Оброблення відсутніх даних

Прикладом набору даних, який потребує попереднього очищення, є набір даних із наявністю значень NaN (Not-A-Number, не число). NaNs використовуються для представлення даних, які не визначені або не можуть бути представлені. Pandas відносять відсутні дані як значення NaN, які також часто називають значеннями NA. NaNs можуть змусити функції аналізу даних різко припинятися під час обчислень, викидати помилки або давати неправильні результати. NaNs також можуть бути призначені цілеспрямовано для рівномірного представлення всіх частин інформації, які відсутні у наборі даних, або неправильних, або нульових значень, або даних, яких просто немає. У багатьох наборах даних відсутні дані, оскільки їх не вдалося зібрати правильно або вони відсутні з самого початку. Ще одна поширена причина NaNs – це перевстановлення даних у наборі даних. Розглянемо приклад.

```
import pandas as pd
import numpy as np
# Create a dataframe with random numbers
dataframe = pd.DataFrame(np.random.randn(3,2), index=['a','c','e'],columns=['one','two'])
dataframe
```

Dataframe with Random Numbers

	one	two
a	-0.610609	0.761838
c	-0.460771	-0.646487

```
# Reindexing the dataframe creates NaNs
dataframe = dataframe.reindex(['a','b','c','d','e'])
dataframe
```

Dataframe with NaN values

	one	two
a	-0.610609	0.761838
b	NaN	NaN
c	-0.460771	-0.646487
d	NaN	NaN

Відсутні значення можуть приймати різні форми на основі типу даних. Типи даних Pandas: об’єкти / рядки, int64 / цілі числа, float64 / floats та datatime64 / timetamps. NaN використовуються для невизначених рядків, цілих та дійсних чисел, а NaT – для часових міток. Можуть також бути ситуації, коли значення Python None буде також представляти відсутні дані. Щоб полегшити виявлення відсутніх значень у наборі даних, Pandas надають функції isnull () та notnull ().

6.2. Перетворення типів даних

Pandas має багато вбудованих функцій для перетворення типів даних. У наступному прикладі набір даних 2 складається з цілих чисел, рядків та чисел типу float. Змінимо стовпець 2 з типу даних об'єкта / рядка на числовий тип даних.

```
import pandas as pd

data2 = [[22,'2017',0.20],[100,'0.33',1.112],[6,'12',0.33]]
df2 = pd.DataFrame(data2, columns = ['col1','col2','col3'])
df2
```

Data2 Data Set

	col1	col2	col3
0	22	2017	0.200
1	100	0.33	1.112
2	6	12	0.330

```
df2.dtypes
```

```
col1     int64
col2    object
col3   float64
dtype: object
```

Функція convert_objects перетворює стовпчик 2 з рядка / об'єкта в числовий тип даних.

```
df2['col2'] = df2['col2'].convert_objects(convert_numeric=True)
df2
```

Column 2 converted from a string/object to a numeric datatype

	col1	col2	col3
0	22	2017.00	0.200
1	100	0.33	1.112
2	6	12.00	0.330

```
df2.dtypes
```

```
col1      int64
col2    float64
col3    float64
dtype: object
```

Стовпець 2 перетворився на тип float64. Це було пов’язано з наявністю рядка "0.33" у стовпці 2. Якби рядок був "33", стовпець перетворився б на цілі числа. Якби рядок був "x" замість "0,33", перетворення призвело б до помилки через неможливість перетворення "x" на числове значення, ціле число або дійсне.

У наступному прикладі дані в стовпці 3 перетворюються з типу даних float64 в об'єктний (рядковий) тип даних. Властивість **dtypes** використовується для перевірки зміни типу даних.

```
df2['col3'] = df2['col3'].astype(str)
df2
```

Data in column 3 is converted from the float64 datatype to the object (string) datatype

	col1	col2	col3
0	22	2017.00	0.2
1	100	0.33	1.112
2	6	12.00	0.33

```
df2.dtypes
```

```
col1      int64
col2    float64
col3     object
dtype: object
```

В наступному прикладі дані в стовпці 1 перетворюються з типу даних int64 в тип даних float64. Властивість **dtypes** використовується для перевірки зміни типу даних.

```
df2['col1'] = df2['col1'].astype(float)
df2
```

Data in column 1 is converted from the int64 datatype to the float64 datatype

	col1	col2	col3
0	22.0	2017.00	0.2
1	100.0	0.33	1.112
2	6.0	12.00	0.33

```
df2.dtypes
```

```
col1     float64
col2     float64
col3      object
dtype: object
```

6.3. Маніпулювання дата фреймами

Очищення набору даних є попереднім завданням перед тим, як здійснювати аналіз даних. Маніпулювання двовимірним фреймом даних за допомогою Pandas у Python може включати видалення, додавання або перейменування стовпців або рядків даних. Для цього потрібно викликати функцію **drop ()**, функцію **loc ()** та функцію **rename ().**

Щоб скинути стовпець даних, викликайте функцію **drop ()**. Створимо простий набір даних. Будемо усувати та додавати стовпці та рядки за допомогою функцій **drop ()** та **add ()**.

```
import pandas as pd
data3 = [[.351,.446,.512],[.112,.980,.122],[.216,.612,.575]]
df3 = pd.DataFrame(data3, columns=['one','two','three'])
df3
```

Simple data set

	one	two	three
0	0.351	0.446	0.512
1	0.112	0.980	0.122

Перший стовпець видалимо за допомогою функції **drop ().**

```
df3.drop(['one'], axis=1, inplace=True)
df3
```

Column one is removed using the drop() function

	two	three
0	0.446	0.512
1	0.980	0.122

Вісь відноситься до стовпців, пронумерованих зліва направо, починаючи зі стовпця індексу на axis=0 і одного стовпця на axis=1.

Такого ж результату можна досягти за допомогою команди **del**.

del df3 [‘one’]

Тепер видалимо перші два рядки (0,1) з наступною функцією **drop ()**, де [0,1] – це рядки, які будуть скинуті, а axis=0 відноситься до стовпця індексу вкрай ліворуч.

```
df3.drop([0,1], axis=0, inplace=True) df3
```

Axis=0 refers to the index column on the far left

	two	three

Додамо стовпець, призначивши йому мітку та значення.

```
df3['four'] =.323
df3
```

Щоб додати рядок, можна скористатися методом **location** або **loc ()**. Метод визначення місцезнаходження також знаходить максимальний індекс або номер останнього рядка, а потім додає до нього 1, створюючи рядок 3.

```
df3.loc[df3.index.max() +1] = [.232,.444,.587]
df3
```

Location method also finds the maximum index or last row number and then adds 1 to it, creating row 3

	two	three	four

Якщо викликати функцію **loc ()** і передати їй номер індексу, вона буде додана як новий нижній рядок. Зверніть увагу, як доданий рядок нумерується 1, хоча це останній рядок.

```
df3.loc[1] =.763
df3
```

Index number passed to the loc() function, it will be appended as a new bottom row

	two	three	four
2	0.612	0.575	0.323

Ми можемо змінити індекс, призначивши нові значення індексу за допомогою властивості **indexframe** dataframe.

```
i = [1,2,3]
df3.index = i
df3
```

New index values assigned with the dataframe index property

	two	three	four
1	0.612	0.575	0.323
2	0.232	0.444	0.587

Поряд із функціями **drop ()** і **loc ()** Pandas також пропонує функцію **rename()** для перейменування мітки стовпців на «один», «два» та «три» відповідно. Для цього потрібно використати функцію **rename()** та призначити стовпці у парі ключ: значення зі старим іменем та новою назвою як пара ключ - значення.

```
df3.rename(columns = {'two':'one','three':'two','four':'three'}, inplace = True)
df3
```

Use the rename function and assign the columns in a key:value pair with the old name and the new name as the key:value pair

	one	two	three
1	0.612	0.575	0.323

Pandas має вбудовані функції для статистичного аналізу набору даних, включаючи функції для обчислення середніх значень, стандартних відхилень та кореляцій. Створимо набір даних та дата фрейм за допомогою масиву даних data4. Фрейм даних виводиться на екран.

```
import pandas as pd
import numpy as np

data4 = [[2,3,5,2,11,3,7,8,10,2,12,7,9,7,4,7,8,12,9,10,6,7]
df4 = pd.DataFrame(data4, columns = ['nums']
df4
```

data4 array of numbers

	Nums
0	2
1	3
2	5
3	4
4	11
5	3
6	7
7	8
8	10
9	2

Середнє значення всіх чисел обчислюється за допомогою методу dataframe **mean ()**. Середнє значення буде 6,863636. Використовуючи крапковий синтаксис, результат також можна округлити до найближчого цілого числа, приєднавши метод **round ()** після **mean ()**.

```
means = df4.mean()
print(means)
```

```
nums 6.863636
dtype: float64
```

```
means = df4.mean().round()
print(means)
```

```
nums 7.0
dtype: float64
```

```
thesum = df.sum()
print(thesum)

thecount = df4.count()
print(thecount)

themeans = (df4.sum()/df4.count()).round()
print(themeans)
```

```
nums   151
dtype: int64
nums   22
dtype: int64
nums   7.0
dtype: float64
```

Якщо в наборі даних є непарна кількість елементів, медіана – це середнє число, відсортоване за числовим порядком. Якщо в наборі даних є парна кількість елементів, медіана обчислюється за допомогою двох середніх чисел.

```
median = df4.median()
print(means)
```

```
nums 7.0
dtype: float64
```

```
thestands = df4.std()
print(thestands)
```

```
nums 3.196657
dtype: float64
```

У другому полі показаний метод **std ()** для обчислення стандартного відхилення. Стандартне відхилення показує величину варіації в наборі значень даних. Низьке стандартне відхилення вказує на те, що числа в наборі даних, як правило, близькі до середнього. Стандартне відхилення знаходимо, беручи квадратний корінь середнього квадратичного відхилення значень від середнього або середнього значення в наборі даних.

Висновок до лекції 6

Найчастіше набори даних, з якими ми працюємо, матимуть несумісність. Очищення даних може включати видалення відсутніх або небажаних значень або зміну формату значень для їх узгодження.

Значення NaN (не число) використовуються для представлення даних, які не визначені або не можуть бути представлені. Pandas посилається на відсутні дані як значення NaN. NaT використовуються для позначок часу. Pandas має багато вбудованих функцій для перетворення типів даних, маніпулювання дата фреймами та проведення статистичного аналізу наборів даних.

Питання для закріплення

1. Як відбувається оброблення відсутніх даних у Pandas?
2. Як відбувається перетворення типів даних?
3. Як відбувається маніпулювання дата фреймами у Pandas?
4. Для чого використовуються функції convert_objects, drop (),loc () та rename ()?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

2. Pandas // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/pandas/pandas_intro.asp

Лекція 7.
Регресійний аналіз даних в Python

План лекції

7.1. Методи та типи аналізу машинного навчання.
7.2. Регресійний аналіз.
7.3. Типи регресійного аналізу.
7.4. Застосування регресійного аналізу.

7.1. Методи та типи аналізу машинного навчання

Машинне навчання вирішує проблеми та можливості, що надаються аналітикою Big Data для моделювання наявних даних, щоб передбачити майбутні результати.

У своїй книзі Кевін Патрік Мерфі визначає машинне навчання як "... сукупність методів, які дозволяють автоматично виявляти шаблони в даних, а потім використовувати ці шаблони для прогнозування майбутніх даних або для виконання інших видів прийняття рішень в умовах невизначеності". Наприклад,

комп'ютерна програма розроблена відеослужбою, щоб рекомендувати фільми окремим користувачам. Алгоритм аналізує фільми, які глядачі вже переглянули, і фільми, які люди з подібними уподобаннями перегляду високо оцінили. Мета – підвищити задоволеність клієнтів відеопослугою.

Методи машинного навчання застосовуються для широкого спектру застосувань, включаючи розпізнавання мови, медичну діагностику, автошколи, рекламні застосунки з рекомендаціями щодо продажу та багато інших.

Незалежно від програми, алгоритми машинного навчання покращують свою ефективність щодо конкретних завдань на основі багаторазового виконання цих завдань, якщо алгоритм і модель здатні впоратися з підвищеною мінливістю, введеною додатковими даними. Це основний тригер пошуку кращих моделей та алгоритмів.

Машинне навчання охоплює багато різних алгоритмів, деякі з широким спектром застосованості, а інші можуть бути придатні для конкретних програм. Ці алгоритми можна розділити на дві основні категорії: контрольовані та без нагляду. Керовані алгоритми машинного навчання – це найчастіше використовувані алгоритми машинного навчання для прогнозної аналітики. Ці алгоритми покладаються на набори даних, які були оброблені людськими експертами (звідси слово "нагляд"). Потім алгоритми дізнаються, як самостійно виконувати ті самі завдання обробки на нових наборах даних. Зокрема, контрольовані методи використовуються для вирішення проблем регресії та класифікації:

Проблеми з регресією – це оцінка математичних зв’язків між неперервними змінними. Цей математичний взаємозв'язок може бути використаний для обчислення значень однієї невідомої змінної з урахуванням відомих значень інших. Прикладами регресії є оцінка положення та швидкості автомобіля за допомогою GPS, прогнозування траєкторії смерчі за допомогою погодних даних або прогнозування майбутньої вартості запасу з використанням історичних даних та інших джерел інформації.

Щоб подумки уявити найпростіший приклад регресії, уявіть дві змінні, значення яких візуалізуються у вигляді точок у двовимірному графіку, подібних до рис. 7.1. Виконання регресії означає пошук лінії, яка найкраще інтерполює значення. Лінія може приймати різних форм і виражається як функція регресії. Функція регресії дозволяє оцінити значення однієї змінної з урахуванням значення іншої для значень, які раніше не спостерігалися.

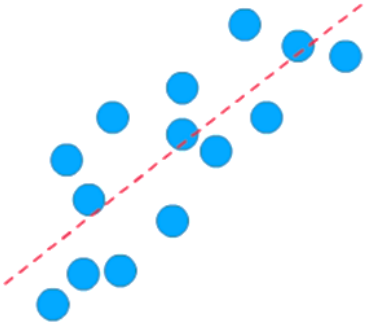


Рис. 7.1. Графічна модель регресії [1]

Проблеми з класифікацією використовуються, коли невідома змінна є дискретною. Зазвичай проблема полягає в оцінці, до якого з набору заздалегідь визначених класів належить конкретний зразок. Типовими прикладами класифікації є розпізнавання зображень або діагностування патологій за допомогою медичних тестів або виявлення облич на знімку.

Візуальну інтерпретацію проблеми класифікації можна побачити у двох вимірах, де точки, що належать до різних класів, позначені різним символом, аналогічним зображенню рис. 7.2.

Алгоритм "вивчає" приклади розташування та форму граничної лінії між класами.

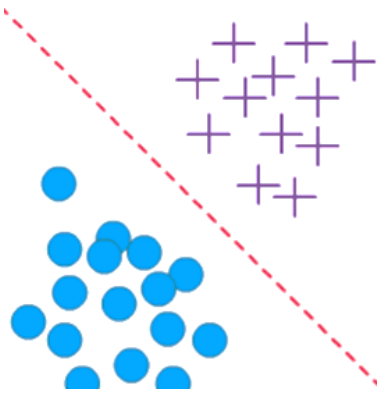


Рис. 7.2. Графічна модель класифікації [1]

7.2. Регресійний аналіз

Регресійний аналіз – один із найдавніших і найбільш часто використовуваних статистичних методів аналізу даних. Основна ідея регресії – це кількісна оцінка математичної залежності між однією або декількома незалежними (також їх називають предиктором) змінною (змінними) і залежною змінною (також її називають цільовою). Регресійний аналіз спирається на набір даних спостережуваних прогнозів і цільових значень. Взаємозв'язок, або функція регресії може бути використана для оцінки значень залежної змінної за межами діапазону спостережуваних значень. Іншими словами, регресійна модель дозволяє аналітику екстраполювати поза наявним набором даних.

Наприклад, при роботі з даними часових рядів, регресія дозволяє аналітику прогнозувати майбутні значення з історичних даних. Регресія шукає зв’язок між будь-якими типами неперервних змінних. Зокрема, вона намагається відповісти на загальне запитання: "на скільки зміниться змінна V_1, якщо змінна (змінні) V_2 (V_3, V_4, V_5) зміниться (зміняться) на величину X?" Простий спосіб візуалізації функції регресії – уявити набір точок у двох вимірах (рис. 7.3).

Частка населення з водійськими правами	Споживання бензину (мільйони галонів)
0,525	541
0,572	524
0,58	561
0,529	414
0,544	410
0,571	457
0,451	344
0,553	467
0,529	464
0,552	498
0,53	580
0,525	471
0,574	525

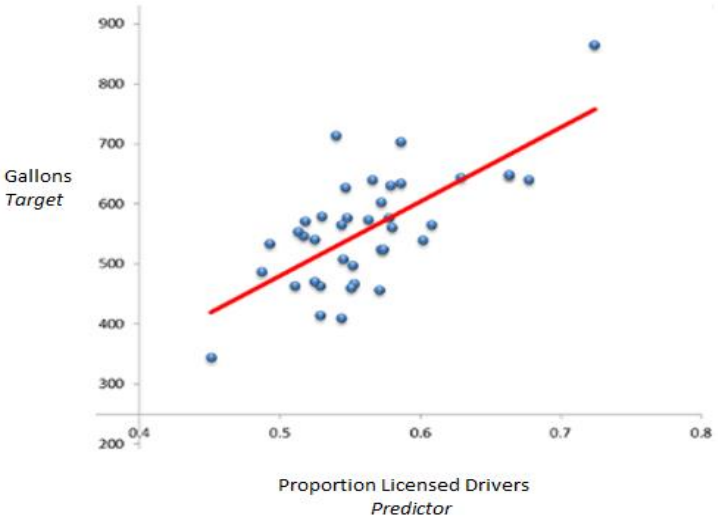


Рис. 7.3. Витрати бензину та відсоток ліцензованих водіїв [1]

Змінна передбачення за умовами, побудованою на осі X, – це частка ліцензованих водіїв у різних географічних районах. На осі Y використовуються значення для цільової змінної, – відповідне споживання бензину. У цьому випадку можлива функція регресії представлена червоною лінією. Той факт, що

в цьому прикладі це пряма лінія, говорить про дуже інтуїтивний результат: збільшення ліцензованих водіїв у цьому районі спричинить пропорційне збільшення споживання бензину. Хоча просте візуальне вивчення розподілу точок даних свідчить про те, що лінія найкраще підходить, регресія не обмежує форму функції регресії.

7.3. Типи регресійного аналізу

Найпоширенішим типом регресії є лінійні регресії. Вони є найпростішими як з обчислювальної, так і з математичної точки зору; і, отже, представляють перший варіант для аналітика даних, представленого проблемою регресії. Незважаючи на назву, лінійна регресія не передбачає встановлення лінії через точки даних. Термін лінійний означає, що функція регресії завжди намагатиметься підходити до даних, використовуючи середньозважене середнє значення інших функцій, будь то лінійна чи ні. Властивість лінійності спрощує обчислення параметрів регресійної моделі, одночасно дозволяючи використовувати практично будь-яку фігуру для відповідності спостереженням. Найпростіший випадок лінійної регресії складається з підгонки до прямої лінії. Це також називається простою лінійною моделлю, як показано на рис.7.4.

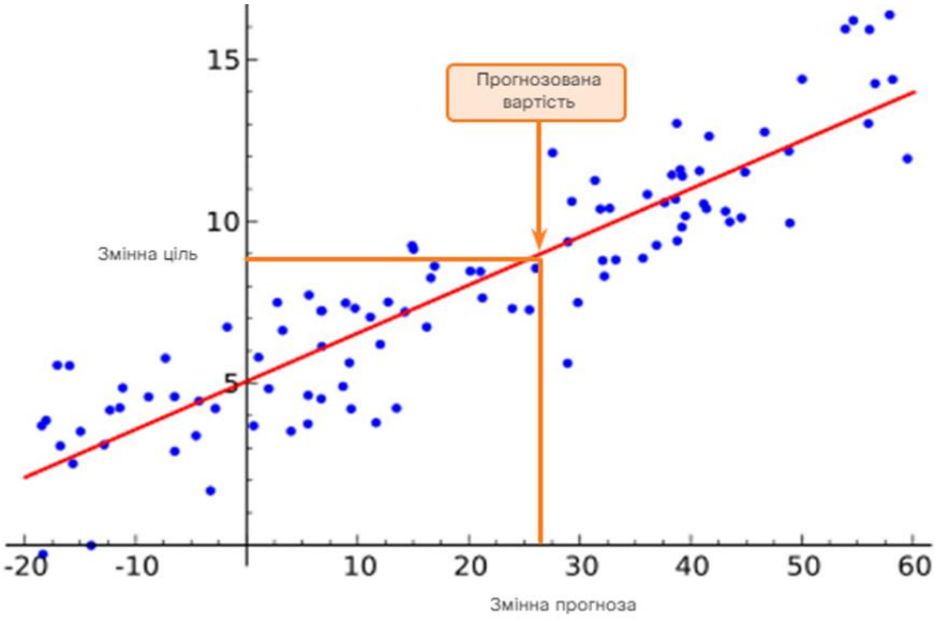
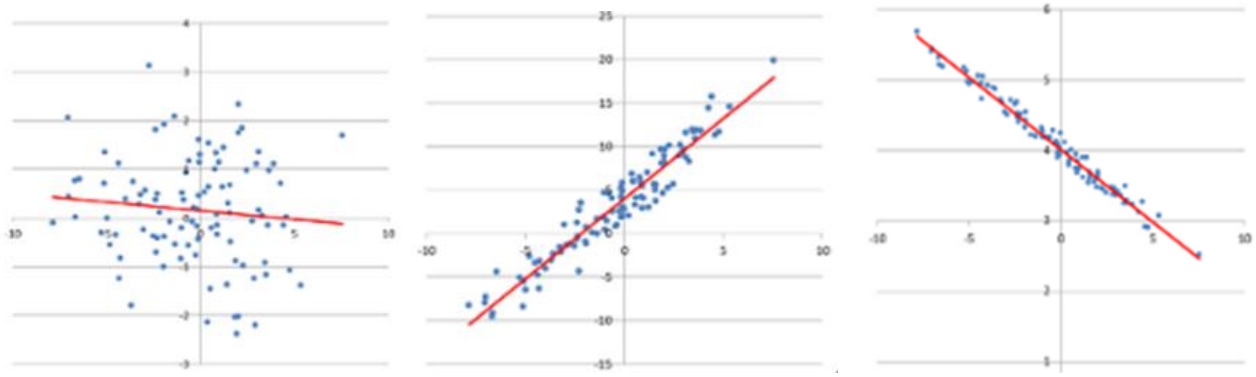


Рис. 7.4. Приклад лінійної регресії [1]

Висока кореляція Пірсона вказує на те, що проста лінійна модель є хорошим кандидатом для відповідності даним (рис. 7.5).



а) r = -0.104 б) r = 0.960 в) r = -0.989

Рис. 7.5. Приклади відсутності кореляції (а), сильних позитивних (б) та негативних корельованих спостережень (в) [1]

Процес регресії в цьому випадку складається з знаходження нахилу та перехоплення лінії, що мінімізує суму відстаней між лінією та всіма точками даних, як показано на рис. 7.6.

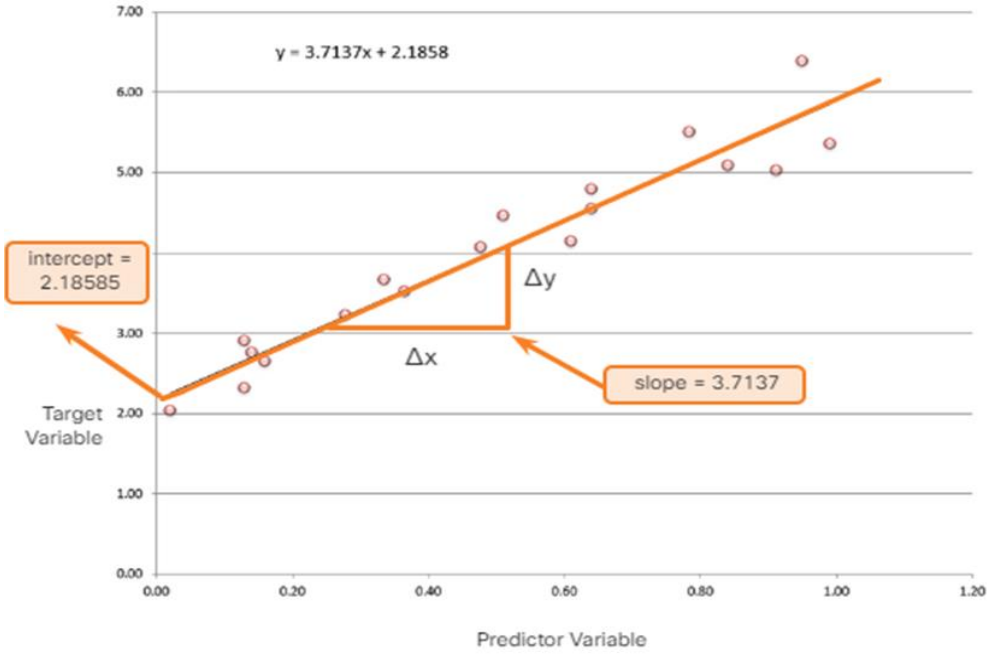


Рис. 7.6. Знаходження нахилу та перетину лінії з віссю Y [1]

При використанні лінійних моделей використовується найпоширеніший алгоритм, що використовується для оцінки цих оптимальних параметрів моделі – метод найменших квадратів (МНК).

На рис. 7.7 ми бачимо три набори даних, у кожному є одна цільова та одна змінна предиктора. У всіх трьох випадках можна спостерігати, як, незважаючи на шум, що впливає на спостереження, існує чітка лінія, яка фіксує основні відносини між змінними. Червона лінія являє собою модель лінійної регресії, яка мінімізує відстань від усіх спостережень. Моделі були отримані за допомогою лінійної регресії.

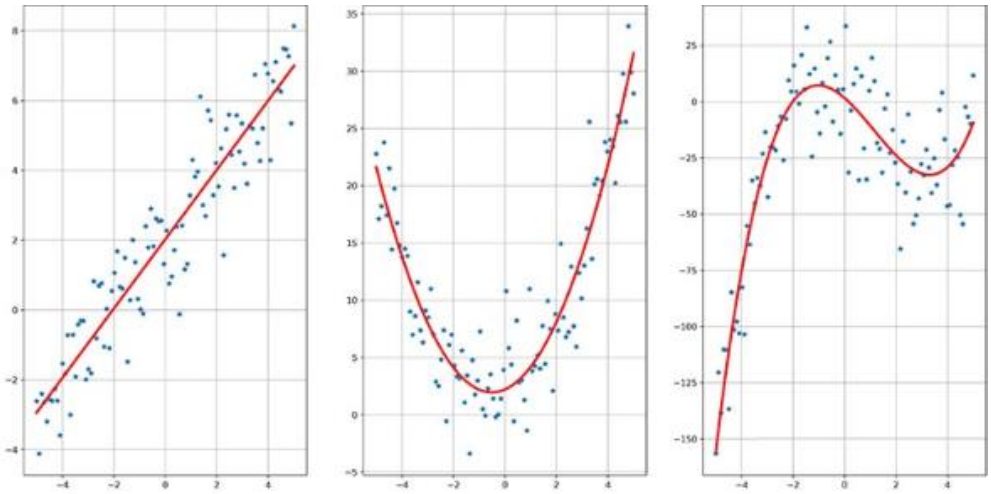


Рис. 7.7. Лінійна та поліноміальні моделі регресії [1]

Розглянемо приклад множинної регресії, коли у нас є не одна, а декілька незалежних змінних. Припустимо, набір даних містить деяку інформацію про автомобілі (рис. 7.8).

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99

Рис. 7.8. Фрагмент набору даних про різні автомобілі
(інформація про модель, об’єм двигуна, масу авто, викиди CO_2) [5]

Ми можемо передбачити викиди CO_2 в автомобілі, виходячи з розміру двигуна та маси автомобіля, щоб зробити прогноз більш точним. Для цього імпортуємо модуль Pandas, який дозволяє читати CSV-файли (у нашому випадку файл cars.csv) та повертати об'єкт DataFrame. Складемо список незалежних значень і викликаємо цю змінну X. Залежні значення поміщаємо у змінну з назвою y. Ми будемо використовувати деякі методи з модуля sklearn, тому нам доведеться також імпортувати цей модуль:

from **sklearn** import linear_model

З модуля sklearn ми будемо використовувати LinearRegression() метод для створення об'єкта лінійної регресії. Цей об'єкт має метод, який називається **fit()** незалежних та залежних значень як параметрів та заповнює об'єкт регресії даними, що описують зв'язок:

regr = linear_model.LinearRegression()

regr.fit(X, y)

Тепер у нас є об'єкт регресії, який готовий прогнозувати значення CO_2 на основі ваги та обсягу автомобіля:

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:

predictedCO2 = regr.predict([[2300, 1300]])

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Результатом виконання програми є число 107.2087328. Тобто ми передбачали, що авто з двигуном об'ємом 1,3 літра і масою 2300 кг буде викидати приблизно 107 грамів CO_2 на кожен пройдений кілометр.

Наступний крок – нам потрібно знайти коефіцієнти нашої регресії. У цьому випадку ми можемо знайти значення коефіцієнта маси авто проти CO_2, а також об'єма двигуна проти CO_2. Відповідь, яку ми отримуємо, говорить нам, що могло б статися, якщо ми збільшимо або зменшимо одне із незалежних значень. Для цього ми використаємо наступний код:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Результат виконання програми:

```
[0,00755095 0,00780526]
```

Масив результатів представляє значення коефіцієнтів для ваги та об'єму.

Вага: 0,00755095

Об'єм: 0,00780526

Ці значення говорять нам, що якщо маса збільшується на 1 кг, викиди CO_2 збільшуються на 0,00755095 г. І якщо розмір двигуна (об'єм) збільшується на 1 $см^3$, викиди CO_2 збільшуються на 0,00780526 г.

Ми вже передбачали, що якщо авто з двигуном 1300 см 3 важить 2300 кг, викиди CO_2 становитимуть приблизно 107 г.

Розглянемо ситуацію, коли ми збільшимо масу авто на 1000 кг (змінимо вагу з 2300 на 3300 кг).

Для цього використаємо наступний код:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Результат виконання програми:

```
[114,75968007]
```

Ми передбачили, що машина з двигуном об’ємом 1,3 літра та масою 3300 кг буде викидати приблизно 115 грамів CO_2 на кожен пройдений кілометр.

Що показує, що коефіцієнт 0,00755095 є правильним:

107,2087328 + (1000 * 0,00755095) = 114,75968 [5].

7.4. Застосування регресійного аналізу

Регресійний аналіз має багато застосувань. Його часто використовують у бізнесі та фінансовому аналізі з історичними даними, щоб повідомити про стратегії подальших дій. Він може бути використаний для прогнозування тенденцій в економіці та може інформувати політичні дії для керівництва економічним зростанням. Також можна прогнозувати поведінку клієнтів, щоб визначити нормальну від шахрайської поведінки у сферах страхування та споживчого кредиту.

У галузі охорони здоров'я може бути використана множинна регресія для оцінки того, яка з ряду змінних може впливати на цільову змінну. Наприклад, взаємозв'язок між групою варіантів способу життя, таких як куріння, кількість фізичних вправ та харчових звичок, можна проаналізувати, щоб визначити, як

вони впливають на змінну стан здоров'я, наприклад, артеріальний тиск, діабет або навіть тривалість життя. Незалежно від програми, будь-яка модель машинного навчання вимагає перевірки. Деякі моделі дуже чутливі до зовнішніх впливів або аномалій даних. Інші моделі можуть генерувати результати, які можуть бути непридатними для відповіді на питання дослідження.

Висновок до лекції 7

Регресійний аналіз – один з найпоширеніших статистичних методів аналізу даних. Основна мета регресії – визначити математичний зв’язок між однією або кількома незалежними змінними та залежною, тобто цільовою змінною. Лінійні регресії є найпростішими як з обчислювальної, так і з математичної точки зору. Термін "лінійний" означає, що функція регресії завжди намагатиметься відповідати даним, використовуючи середньозважене значення інших функцій, незалежно від того, є ці функції лінійними чи ні. Регресійний аналіз часто використовується у фінансовому та бізнес-аналізі для формування стратегій подальших дій. Він може бути використаний для прогнозування економічних тенденцій та для управління економічним зростанням.

Питання для закріплення

1. Які методи та типи аналізу машинного навчання ви знаєте?
2. Для чого використовується регресійний аналіз?
3. Які типи регресійного аналізу ви знаєте?
4. Які застосування регресійного аналізу в реальному житті ви знаєте? Наведіть приклади.

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
2. Machine Learning // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/python_ml_getting_started.asp
3. Linear Regression // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/python_ml_linear_regression.asp
4. Polynomial Regression // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/python_ml_polynomial_regression.asp
5. Multiple Regression // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/python_ml_multiple_regression.asp

Лекція 8.
Помилки в аналізі даних та прогнозній аналітиці. Оцінка помилок регресії засобами Python. Призначення бібліотеки scikit-learn

План лекції

8.1 Помилки в аналізі даних та прогнозній аналітиці.
8.2. Оцінка помилок регресії засобами Python.
8.3. Призначення бібліотеки scikit-learn.

8.1 Помилки в аналізі даних та прогнозній аналітиці

Помилки та невизначеність впливають на процес аналізу даних на різних рівнях. Перший тип помилок – це *похибка вимірювання*. Дані потрібно очищати, оскільки значення змінних можуть бути пошкоджені шумом. Але звідки цей шум? Часто помилка викликається датчиком, або людиною під час читання чи використанням датчика. Будь-який пристрій для проведення вимірювань обмежений у своїй точності. Тому всі вимірювання мають вбудований компонент помилки. Пристрій завжди матиме вбудовану помилку. Наприклад, вимірювальна стрічка, яка використовується для маркування лінії різання фанери на шматки, має вбудовану помилку вимірювання. Помилка в кілька міліметрів не вплине на ефективність штормових заслінок. Однак, наприклад, потрібна більша точність при нарізанні матеріалів для кухонних шаф. Через помилку вимірювання справжнє значення не може бути відоме, але ця помилка може бути вивчена статистично і врахована і визначається як різниця між справжнім значенням (яке невідомо) і вимірюваним.

Інший тип помилки – це *похибка передбачення*. При контрольованому навчанні похибка прогнозування кількісно визначається як різниця між значенням, передбаченим моделлю, і спостережуваним значенням. На спостережуване значення впливає похибка вимірювання, і хоча це неможливо усунути, існують методи, такі як перехресна перевірка, щоб обмежити його ефекти.

Грубі помилки – спричинені помилкою в приладі, який використовується для проведення вимірювання, або в записі результату вимірювання. Наприклад, спостерігач записує 1,10 замість фактичного вимірювання 1,01.

Випадкові помилки – спричинені чинниками, які випадковим чином впливають на вимірювання даних. Наприклад, калібрована шкала у продуктовому магазині може мати плюс або мінус помилку в 1 грам кожного разу, коли ви зважуєте один і той же предмет.

Систематичні помилки – спричинені інструментальними чи екологічними факторами, які впливають на усі вимірювання, проведені протягом певного періоду часу. Наприклад, не калібрована шкала буде генерувати систематичну помилку щоразу, коли проводиться вимірювання. Випадкові помилки, як правило, створюють нормальний розподіл навколо середнього спостереження (рис. 8.1). Можна побудувати статистичну модель помилки, в цьому випадку алгоритми регресії можуть легко її врахувати. Для деяких методів факт, що помилка слідує за нормальним розподілом, є вимогою.

Систематичні помилки зміщують розподіл спостережень (рис. 8.1) в ту чи іншу сторону. Тому систематичну помилку важче усунути, оскільки справжнє значення невідоме, єдиний спосіб виявити систематичну помилку – це використовувати іншу систему вимірювань, яку ми вважаємо більш надійною.

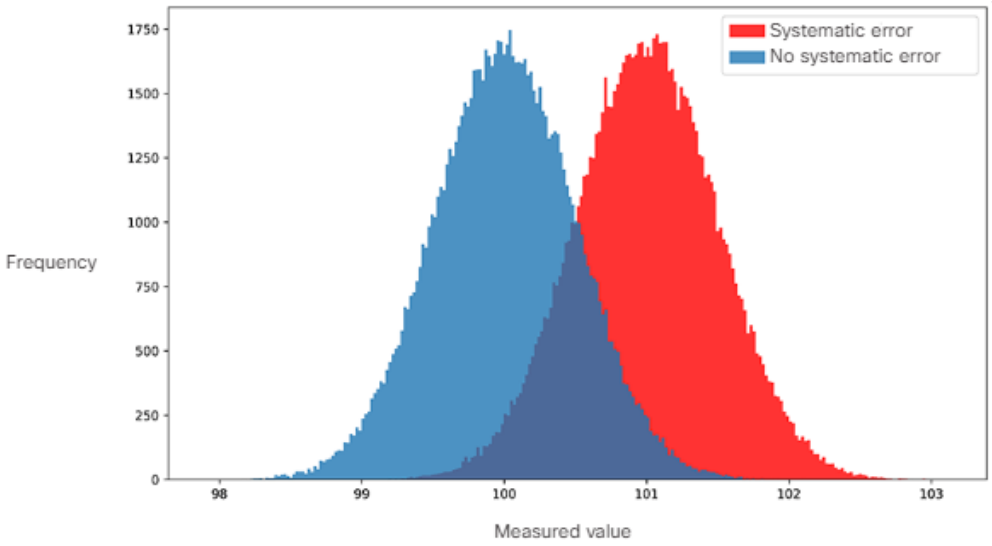


Рис. 8.1. Розподіл спостережень для систематичних та випадкових помилок [1]

Похибка прогнозування – це різниця між значенням, передбаченим регресійною або класифікаційною моделлю, і вимірюваним значенням.

Похибка прогнозування – це відстань між функцією регресії та точками даних. Зокрема, прийнято оцінювати помилку прогнозування, використовуючи середнє значення суми відстаней у квадраті для всіх точок.

При регресії помилка на навчальному наборі менша, ніж помилка на наборі перевірки. Помилка передбачення має два компоненти. Перший компонент обумовлений вибором моделі. Незалежно від алгоритму, щоразу, коли ми підходимо до регресії або класифікаційної моделі, ми робимо припущення про те, як розподіляються дані, що неминуче є наближенням. Наприклад, ми могли б підігнати модель, яка є гарним наближенням лише для заданого діапазону зразків, але не вдається зафіксувати взаємозв'язок поза нею. Девіз аналітиків даних: "Кожна модель помилкова, але деякі з них корисні". На рис. 8.2 показана різниця між поліноміальною моделлю другого порядку, на рис. 8.3 показана різниця між моделлю третього порядку.

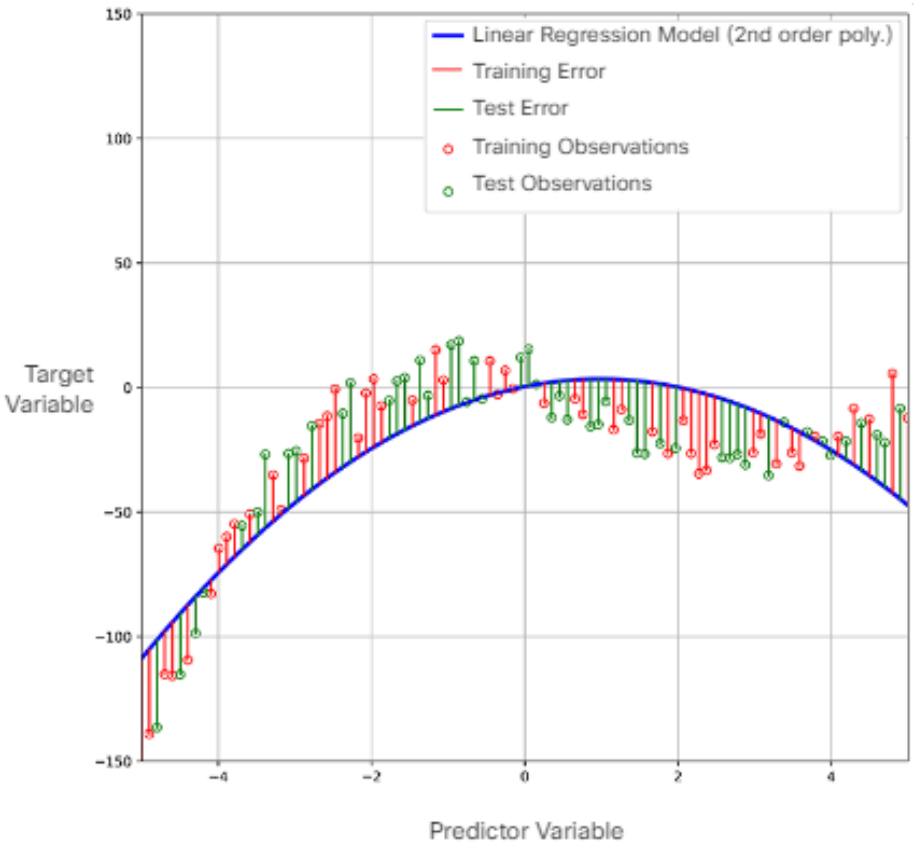


Рис. 8.2. Лінійна регресійна модель другого порядку [1]

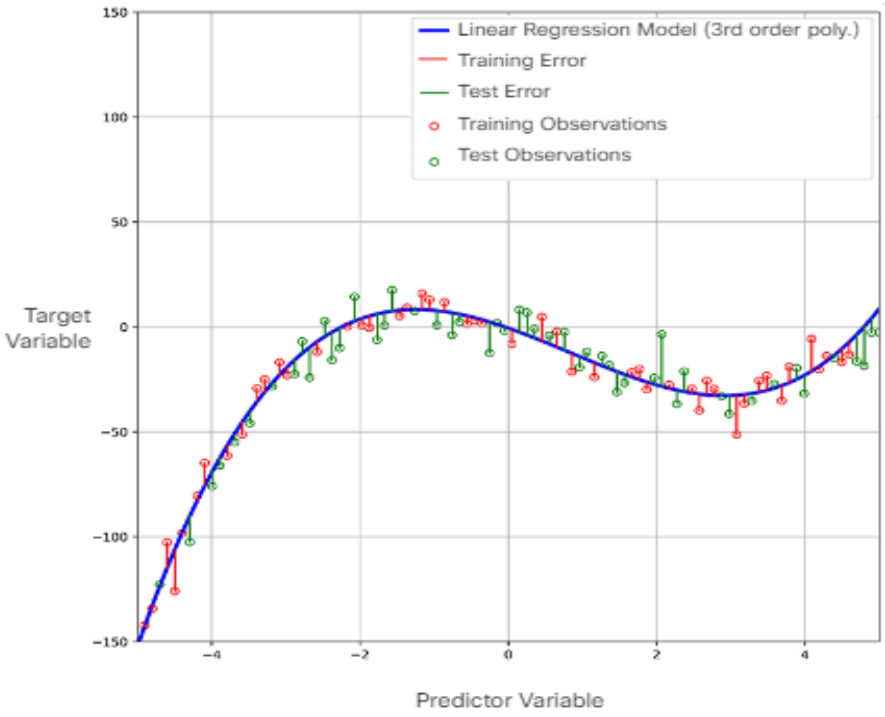


Рис. 8.3. Лінійна регресійна модель третього порядку [1]

Модель третього порядку, безумовно, краще працює в плані помилок (рис. 8.4).

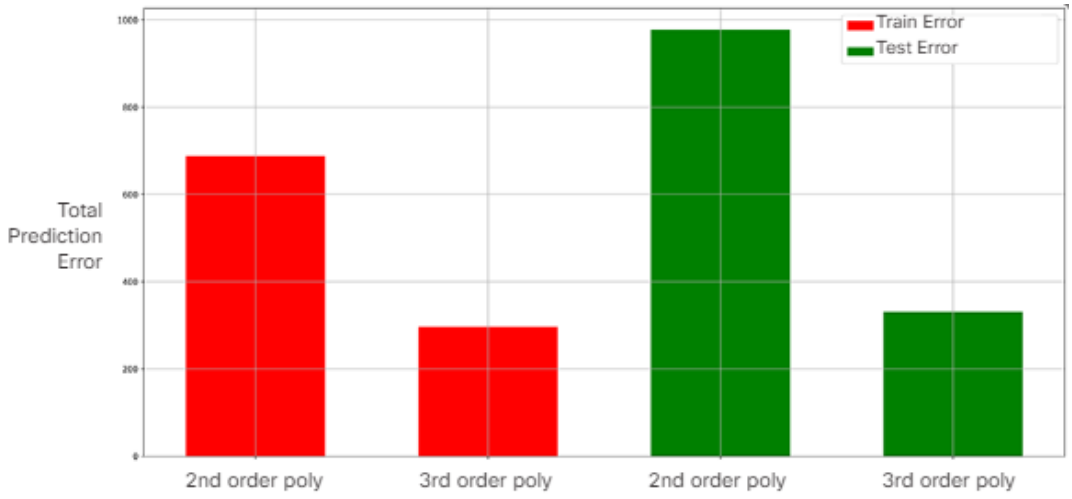


Рис. 8.3. Помилки прогнозу для лінійних регресійних модельдругого та третього порядку [1]

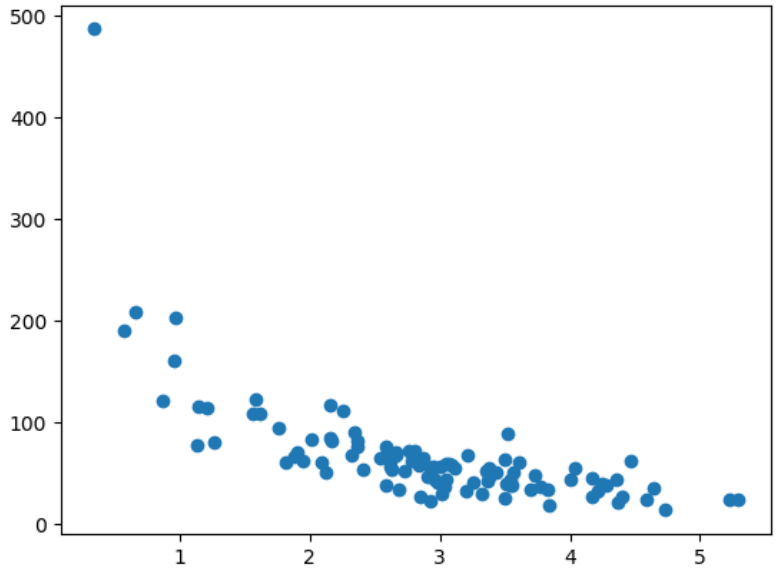
Навіть коли обрана модель чудово відображає справжній розподіл, все ж таки буде різниця між передбачуваними та фактичними значеннями через помилку вимірювання. Це неможливо усунути; а отже, похибка вимірювання впливає на регресійну модель.

8.2. Оцінка помилок регресії засобами Python

Метод вимірювання точності моделі має назву Train / Test, оскільки ми розділяємо дані на два набори: навчальний набір та набір для тестування (80% даних для навчання та 20% для тестування). Ми тренуємо модель за допомогою навчального набору та перевіряємо модель з допомогою набору для тестування. Навчити модель означає створити модель. Перевірка моделі означає перевірку точності моделі. Наш набір даних ілюструє 100 покупців у магазині та їхні звички покупок залежно від часу перебування у магазині.

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.show()
```

Результат:



Вісь x представляє кількість хвилин до здійснення покупки.

Вісь y представляє суму грошей, витрачених на покупку.

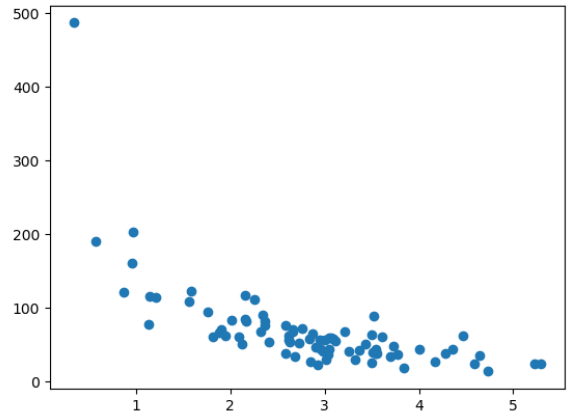
Набір тренувань повинен бути випадковим відбором 80% вихідних даних. Набір для тестування повинен складати решту 20%.

```
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

Відобразимо той самий графік розсіювання з навчальним набором:

```
plt.scatter(train_x, train_y)
plt.show()
```

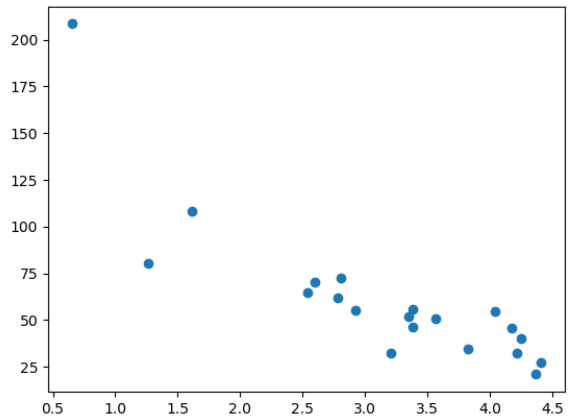
Результат:



Щоб переконатися, що набір для тестування не зовсім відрізняється, ми також поглянемо на комплект для тестування.

```
plt.scatter(test_x, test_y)
plt.show()
```

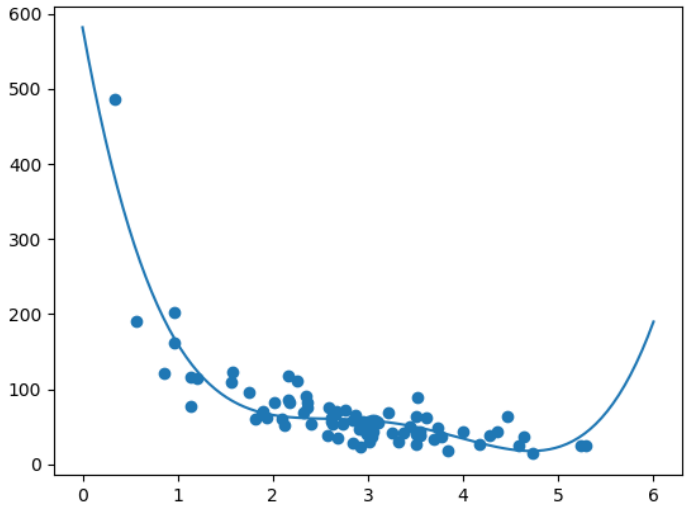
Результат: тестовий набір також виглядає як оригінальний набір даних:



Скоріше за все, найкращим варіантом буде поліноміальна регресія, тому проведемо лінію поліноміальної регресії через точки даних. Щоб провести лінію через точки даних, ми використовуємо **метод plot()** модуля **matplotlib**:

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Результат:



Результат може допомогти нам передбачити значення поза набором даних. Приклад: рядок вказує, що клієнт, який проводить у магазині 6 хвилин, скоріш за все, зробить покупку вартістю 200 $.

Оцінка R-квадрат є хорошим показником того, наскільки набір даних відповідає моделі. R-квадрат вимірює взаємозв'язок між віссю x та віссю y, і значення коливається від 0 до 1, де 0 означає відсутність зв'язку, а 1 означає повністю взаємопов'язаність. Модуль **sklearn** містить метод **r2_score(),** що допоможе нам знайти цей взаємозв'язок.

Виміряємо взаємозв'язок між хвилинами перебування клієнта в магазині та кількістю витрачених грошей. Перевіримо, наскільки добре навчальні дані відповідають поліноміальній регресії.

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(train_y, mymodel(train_x))

print(r2)
```

Результат 0,799 показує, що існує досить хороший зв’язок [2]. Зараз ми створили модель, яка нормальна, принаймні, коли справа стосується навчальних даних. Тепер ми хочемо протестувати модель із даними тестування, щоб побачити, чи дає нам той самий результат.

Знайдемо оцінку R2 при використанні даних тестування:

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(test_y, mymodel(test_x))

print(r2)
```

Результат 0.809 показує, що модель відповідає тестовому набору, і ми впевнені, що можемо використовувати модель для прогнозування майбутніх значень. Тепер, коли ми встановили, що з нашою моделлю все в порядку, ми можемо почати прогнозувати нові значення. Визначимо, скільки грошей витратить покупець, якщо він або він пробуде в магазині 5 хвилин, для цього ми використовуємо команду print(mymodel(5)):

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

print(mymodel(5))
```

Результат:

```
22.87962591812061
```

Ми передбачили, що клієнт може витратити 22,88 доларів, що відповідає відповідній точці на діаграмі:

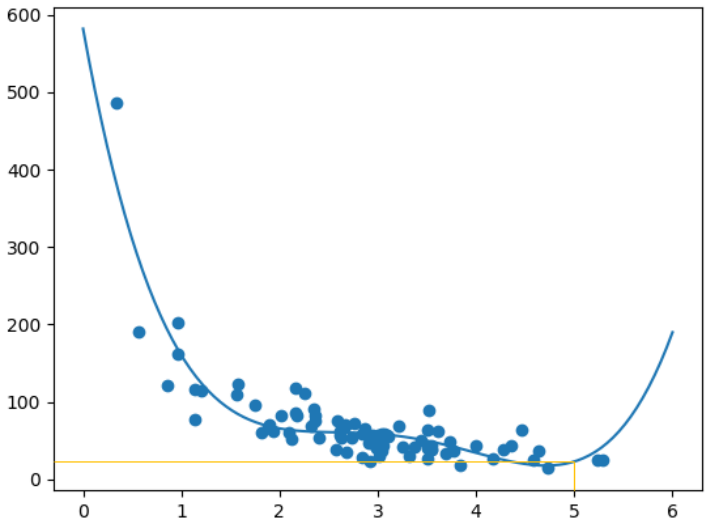


Рис. 8.4. Передбачення витрат клієнта відносно часу перебування в магазині [2]

8.3. Призначення бібліотеки scikit-learn

Scikit-learn – це безкоштовна бібліотека машинного навчання для мови програмування Python. Має різні алгоритми класифікації, регресії та кластеризації, включаючи SVM (support vector machines), дерева рішень, k-середні значення та DBSCAN (Density-Based Spatial Clustering of Applications with Noise), і призначений для взаємодії з числовими та науковими бібліотеками Python NumPy та SciPy.

Scikit-learn займається вивченням інформації з одного або декількох наборів даних, які представлені у вигляді 2D-масивів. Їх можна зрозуміти як перелік багатовимірних спостережень.

Основний API, реалізований scikit-learn, – це оцінювач. Оцінювач – це будь-який об'єкт, який вивчає дані; це може бути алгоритм класифікації, регресії або кластеризації або трансформатор, який витягує / фільтрує корисні функції з необроблених даних. Усі об'єкти оцінювача мають метод підгонки, який використовує набір даних (зазвичай це 2D-масив) [3].

Висновок до лекції 8

Похибки та невизначеність впливають на процес аналізу даних на різних рівнях. Перший тип похибки – це похибка вимірювання. Інший тип похибки – це похибка прогнозування. При контрольованому навчанні похибка прогнозування визначається кількісно як різниця між величиною, передбаченою моделлю, та спостережуваною величиною. Грубі помилки – спричинені помилкою в приладі, який використовується для вимірювання, або в записі результату вимірювання. Випадкові помилки – спричинені факторами, які випадково впливають на вимірювання на вибірці даних.

Систематичні помилки – спричинені інструментальними або екологічними факторами, які впливають на всі вимірювання, проведені протягом певного періоду часу. Випадкові помилки, як правило, створюють нормальний розподіл навколо середньої величини спостереження.

Помилка передбачення – це відстань між функцією регресії та точками даних. Помилка передбачення має дві складові. Перший компонент обумовлений вибором моделі, ми робимо припущення про те, як розподіляються дані, що неминуче є наближенням. Навіть коли обрана модель ідеально відображає справжній розподіл, все одно існуватимуть відмінності між прогнозованими та фактичними значеннями через похибку вимірювання.

Питання для закріплення

1. Які типи помилок в аналізі даних та прогнозній аналітиці ви знаєте?
2. Як відбувається оцінка помилок регресії засобами Python?
3. Яке призначення та можливості бібліотеки scikit-learn?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
2. Machine Learning - Train/Test // Електронний ресурс. Режим доступу: https://www.w3schools.com/python/python_ml_train_test.asp
3. An introduction to machine learning with scikit-learn // Електронний ресурс. Режим доступу: https://scikit-learn.org/stable/tutorial/basic/tutorial.html

Лекція 9.

Алгоритми класифікації даних. Застосування та проблеми класифікацій. Модель класифікатора дерева рішень

План лекції

9.1. Проблеми класифікації.
9.2. Алгоритми класифікації.
9.3. Візуалізація класифікацій.
9.4. Застосування та валідація класифікацій.

9.1. Проблеми класифікації

Класифікація – це розповсюджена проблема машинного навчання, яка входить до категорії контрольованого навчання. За останнє десятиліття було досягнуто значних покращень, особливо в області розпізнавання зображень. Класифікація може розглядатися як проблема регресії, коли цільова змінна є дискретною, і представляє клас, в якому експерт-людина класифікує вибірку

даних. У задачах класифікації прийнято надавати не лише набір прикладів точок даних кожного класу, а й також встановлювати, які особливості кожної точки даних є більш корисними для оцінки відповідного класу. Ці функції можуть бути доступними з датчиків, але частіше їх потрібно обчислювати (або витягувати) із вихідних даних, перш ніж надавати в алгоритм навчання. Визначення відповідних функцій є вирішальним кроком, який, за винятком дуже вдосконалених алгоритмів, таких як глибоке навчання, спирається на знання експертів людини. Наприклад, туристична компанія зацікавлена в наданні рейтингу надійності рейсів, які вона знаходить для клієнтів. Шляхом пробної помилки різних моделей було визначено, які змінні серед усіх наборів даних є найбільш релевантними для класифікацій. Це також відомо як змінні з найбільшою дискримінантною силою. Тільки ці відповідні ознаки витягуються з даних і використовуються для навчання класифікатора. Компанія вирішила використати класифікатор для прогнозування того, які рейси найімовірніше належать до груп рейсів, які проводяться вчасно, із запізненням або скасованими рейсами. Шляхом пробної помилки різних моделей було встановлено, які змінні серед усіх, що містяться в наборі даних, є найбільш релевантними для класифікацій (також, як кажуть, мають найбільшу дискримінантну силу). Тільки ці відповідні функції витягуються з даних і використовуються для підготовки класифікатора. Рейтинг надійності призначений для повідомлення про ступінь ймовірності того, що рейс буде виконаний вчасно, затриманий або скасований. Туристична компанія має доступ до великої кількості даних про різні авіакомпанії, рейси, походження та пункти призначення, статус рейсу та іншу інформацію.

9.2. Алгоритми класифікації

Розглянемо деякі алгоритми класифікаторів, які популярні для різних цілей.

Алгоритм k-найближчого сусіда (k-nearest neighbors algorithm, k-NN) – можливо, найпростіший класифікатор, який використовує відстань між зразками навчання як міру подібності.

Щоб уявити, як працює класифікатор k-NN, уявіть, що кожен зразок має дві особливості, для яких значення можуть бути представлені на двовимірному графіку. На рис. 9.1 точки даних кожного класу позначені різними символами. Відстань між точками представляє різницю між значеннями ознак. З огляду на нову точку даних, класифікатор k-NN розгляне найближчі точки навчання. Прогнозований клас для нової точки буде найпоширенішим класом серед k сусідів.

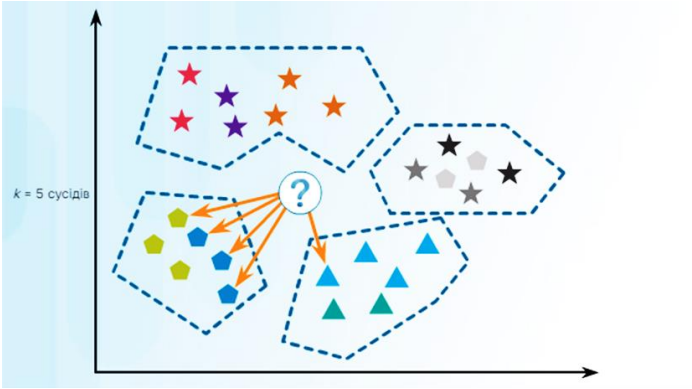


Рис. 9.1. Метод k-найближчих сусідів (k-NN) [1]

Метод опорних векторів (Support vector machines, SVM) – приклад керованих класифікаторів машинного навчання. Замість того, щоб базувати належність до категорії на відстані від інших точок, опорні векторні машини обчислюють кордон або гіперплощину, що краще розділяє групи. На рис. 9.2. Н3 – це гіперплощина, яка максимально збільшує відстань між навчальними точками двох класів (позначено кольором або чорно-білим кольором). Коли буде представлена нова точка даних, вона буде класифікована на основі того, лежить вона на одній або іншій стороні від Н3.

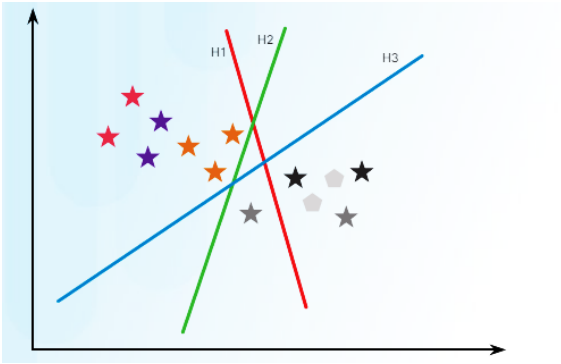


Рис. 9.2. Метод опорних векторів [1]

Дерева рішень представляють проблему класифікації як сукупність рішень на основі значень ознак. Кожен вузол дерева представляє поріг над значенням функції та розбиває навчальні зразки на два менші набори (рис. 9.3.).

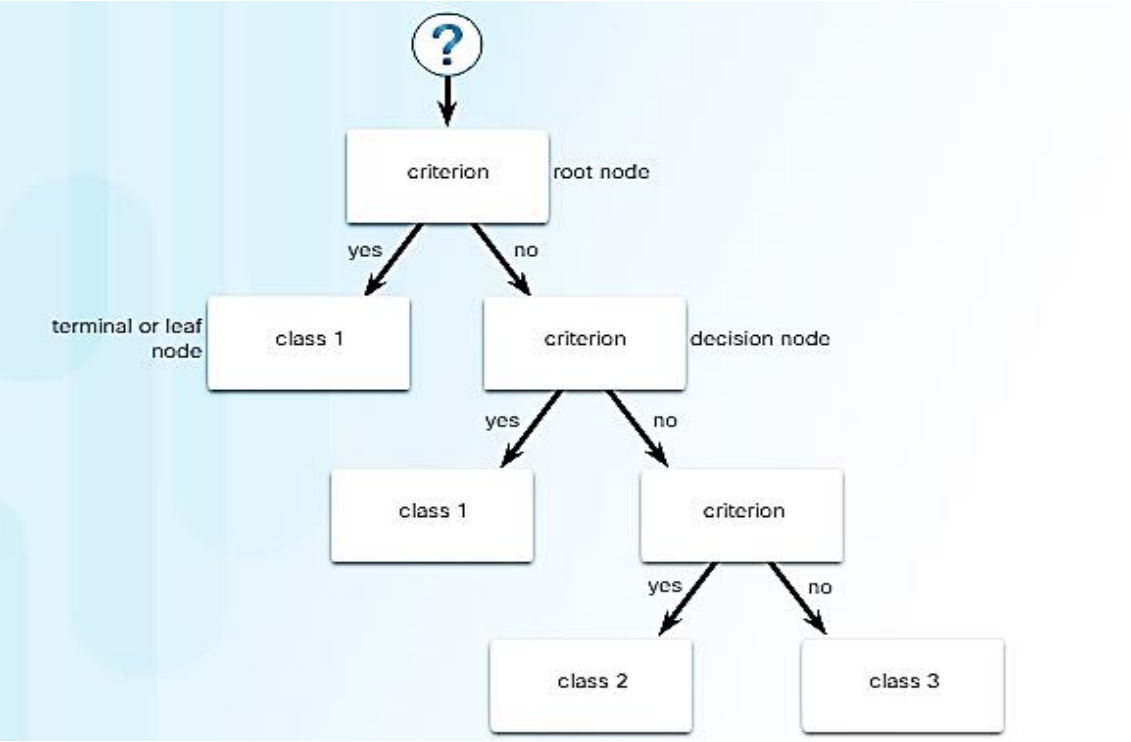


Рис. 9.3. Спрощений вигляд дерева двійкових рішень та типів вузлів [1]

Процес прийняття рішення повторюється за всіма ознаками, вирощуючи дерево до тих пір, поки не буде обчислений оптимальний спосіб поділу зразків. Класифікацію нового зразка можна отримати, слідуючи гілкам дерев на основі значень його ознак.

9.3. Візуалізація класифікацій

Різні типи візуалізації покращують дослідницьке використання алгоритмів класифікації. На рис. 9.4. зображена візуалізація для k-NN-аналізу, який присвоює нові спостереження одному з трьох класів. Спостереження класифікуються на основі членства 15 найближчих сусідів.

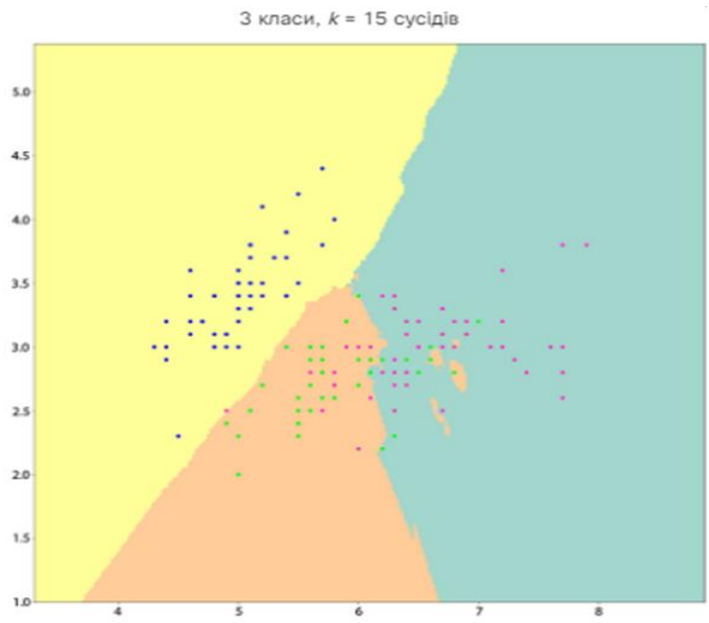


Рис. 9.4. Візуалізація для k-NN-аналізу [1]

На рис. 9.5. зображена дуже проста візуалізація дерева рішень для класифікатора, побудованого для передбачення, які з пасажирів на затопленому круїзному кораблі «Титанік» будуть вцілілими чи жертвами. Вузли дерева рішень включають міру ймовірності та відсоток пасажирів, який представлений кожним вузлом. Це дерево рішень дуже корисне при виявленні факторів, таких як стать, які мали найбільший вплив на життєздатність. Ця система може бути забезпечена вигаданими пасажирами, і вона дуже точно класифікувала б нових пасажирів за результатами виживання.

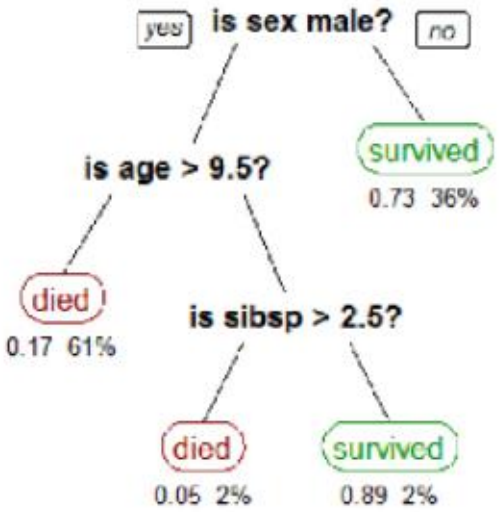


Рис. 9.5. Візуалізація простого бінарного дерева рішень [1]

На рис. 9.6. зображено тривимірний сюжет методу SVM. У цьому випадку гіперплощина, яка розділяє дві групи, отримується з трьох змінних для кожного спостереження. Існує невелика кількість помилок, як показано в точках даних, які з'являються на неправильній стороні гіперплощини.

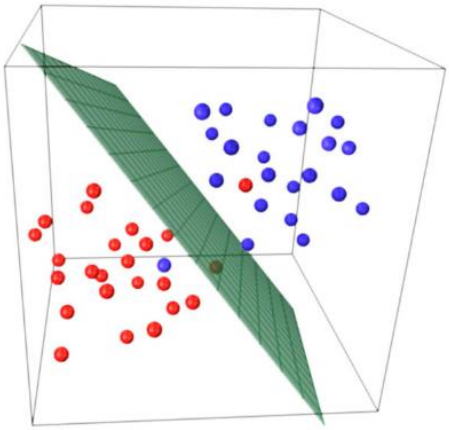


Рис. 9.6. Візуалізація методу опорних векторів у тривимірному просторі даних [1]

9.4. Застосування та валідація класифікацій

Розглянемо приклади застосування алгоритмів класифікації.

Оцінка ризиків – класифікаційні системи можна використовувати, щоб визначити, який із багатьох факторів сприяє ймовірності різних ризиків. Наприклад, ряд факторів може бути використаний для класифікації користувачів автомобільного страхування на категорії низького, середнього та високого ризику та коригування премій, які сплачують водії відповідно до рівня ризику.

Медична діагностика – системи класифікації можуть використовувати керовані питання для побудови дерева рішень, які можуть допомогти діагностувати різні захворювання та ризики захворювань. Системи класифікації машинного навчання можуть також проводити попередній аналіз великої кількості діагностичних зображень та встановлювати підозрілі умови для огляду лікарями.

Розпізнавання зображень – наприклад, для розпізнавання рукописного тексту система може працювати над завданням ідентифікації рукописних

цифр. Числа 0 - 9 можна вважати класами. Класифікатор забезпечений великою вибіркою рукописних цифр, кожна з яких позначена фактично представленою цифрою. Класифікатор шукатиме функції, які, швидше за все, будуть присутніми та унікальними для кожної з цифр.

Наукові відкриття часто походять від використання наукового методу, який містить наступні кроки.

Крок 1. Задайте запитання щодо спостереження, наприклад, що, коли, як чи чому?

Крок 2. Виконайте дослідження.

Крок 3. Сформуйте гіпотезу дослідження.

Крок 4. Перевірте гіпотезу шляхом експериментів.

Крок 5. Проаналізуйте дані експериментів, щоб зробити висновок.

Крок 6. Повідомте результати процесу.

Питання валідності та надійності є фундаментальними для підтримки результатів, заявлених будь-яким експериментом чи дослідженням.

Дослідники розрізняють чотири типи надійності:

Міжрейтингова надійність – наскільки аналогічно оцінюють різні люди на одному тесті?

Надійність тестування-повторне тестування – скільки варіацій між балами для однієї і тієї ж людини, яка приймає тест кілька разів?

Надійність паралельних форм – наскільки схожі результати двох різних тестів, побудованих з одного вмісту?

Надійність внутрішньої узгодженості – яка різниця результатів для різних предметів одного і того ж тесту?

Однак в аналітиці даних повторення експерименту може бути занадто дорогим або навіть неможливим. Прикладом є система класифікації зображень, реалізована Facebook, яка дозволяє користувачам шукати зображення за допомогою текстового опису. Розробляючи таке рішення, аналітики із даних Facebook отримують доступ до мільйонів картинок із текстовим описом, наданим людськими експертами. Вони можуть чітко налаштувати свій алгоритм

класифікації, щоб поліпшити його продуктивність, але вони не можуть знати, як він буде вести себе з новими картинками, які користувачі розмістять у майбутньому, і не можуть оцінити, чи дасть це правильну відповідь чи ні. Тож як вони можуть бути впевнені, що система класифікації працюватиме із зображеннями, які вона раніше не обробляла? Вони вдаються до методу, званого перехресною валідацією, коли ви тренуєте свій алгоритм, використовуючи лише випадково вибраний зразок даних, який називається навчальним набором.

Потім модель оцінюється за рештою даних, що називається набором валідації. Ефективність класифікації, яку отримує класифікаційна система на навчальному наборі, зазвичай вища, ніж у набору валідацій. Однак це краще відображає, як алгоритм поводиться із зразками, які він раніше не обробляв.

Успіх алгоритму класифікації з використанням зображень, які користувач опублікує в майбутньому, буде залежати від того, наскільки навчальний набір представляв весь набір даних. Якщо, наприклад, користувачі почнуть публікувати зображення сонячного затемнення, система знатиме, як класифікувати його, лише якщо у навчальному наборі були приклади сонячного затемнення.

Рішення для аналітики даних часом піддаються однаковим упередженням людей, оскільки це зміщення виражається в наборах даних, що використовуються для їх навчання.

Висновок до лекції 9

Класифікація може розглядатися як проблема регресії, коли цільова змінна є дискретною, і представляє клас, в якому експерт-людина класифікував вибірку даних. Основними алгоритмами класифікації є методи k-NN-аналізу, SVM та дерева рішень.

Алгоритми класифікації мають багато застосувань. Наприклад, вони використовуються для оцінка ризику, для визначення того, який із багатьох факторів сприяє ймовірності різних ризиків.

Медична діагностика – системи класифікації можуть використовувати керовані запитання для побудови дерева рішень, яке може допомогти діагностувати різні захворювання та ризики захворювань.

Розпізнавання зображень – при розпізнаванні рукописного вводу система може працювати над завданням ідентифікації рукописних цифр.

Питання для закріплення

1. Що таке класифікація даних?
2. Які алгоритми класифікацій ви знаєте?
3. Яке застосування класифікацій?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
2. Cluster analysis // Електронний ресурс. Режим доступу: https://en.wikipedia.org/wiki/Cluster_analysis
3. Support-vector machine // Електронний ресурс. Режим доступу: https://en.wikipedia.org/wiki/Support-vector_machine
4. Decision tree// Електронний ресурс. Режим доступу: https://en.wikipedia.org/wiki/Decision_tree

Лекція 10.

Модуль Pyplot. Інструмент Plotly. Типи візуалізації даних. Візуалізація аномалій. Використання бібліотек Folium та Leaflet.js для побудови карт

План лекції

10.1. Модуль Pyplot.
10.2. Інструмент Plotly.
10.3. Типи візуалізації даних.
10.4. Візуалізація аномалій.
10.5. Використання бібліотек Folium та Leaflet.js для побудови карт.

10.1. Модуль Pyplot

Pyplot – це модуль **matplotlib**, який включає набір функцій стилю, які можна використовувати для створення та налаштування сюжету.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1,2,3,4], [1, 4, 9, 16])
plt.plot([1,2,3,4], [1, 5.7, 15.6, 32])
plt.plot([1,2,3,4], [1, 8, 27, 48])
plt.plot([1,2,3,4], [1, 11.3, 46.8, 128])
plt.xlabel('X Label Here')
plt.ylabel('Y Label Here')
plt.title('Title Here')
plt.show()
```

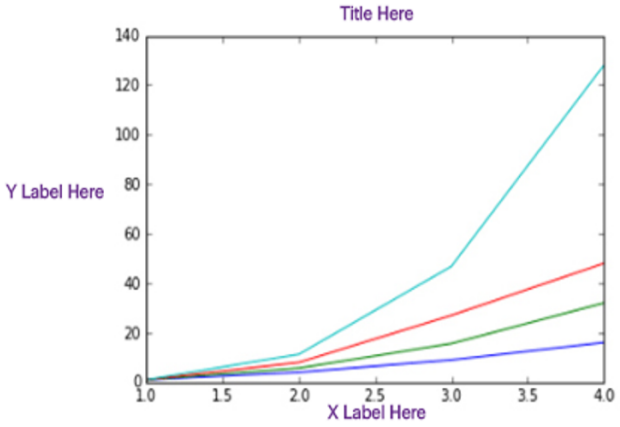


Рис. 10.1. Створення швидкого сюжету із типовими стилями [1]

Ви можете налаштувати стиль за замовчуванням, додавши вбудований код. На рис. 10.2 реалізовані наступні модифікації:

• Стандартний розмір цифри – 6,4 на 4,8 дюйма. Щоб змінити розмір, використовуйте код plt.figure (figsize = (ширина, висота).

• Ширина лінії за замовчуванням до 1,5 балів. Щоб змінити розмір, додайте атрибут linewidth.

• За замовчуванням розмір шрифту становить 10,0 балів. Щоб збільшити розмір шрифту, додайте атрибут fontsize.

```
plt.figure(figsize = (9.6,7.2))
plt.plot([1,2,3,4], [1, 4, 9, 16])
plt.plot([1,2,3,4], [1, 5.7, 15.6, 32])
plt.plot([1,2,3,4], [1, 8, 27, 48])
plt.plot([1,2,3,4], [1, 11.3, 46.8, 128], linewidth = 3.0)
plt.xlabel('X Label Here', fontsize = 14)
plt.ylabel('Y Label Here', fontsize = 14)
plt.title('Title Here', fontsize = 20)
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10)
plt.show()
```

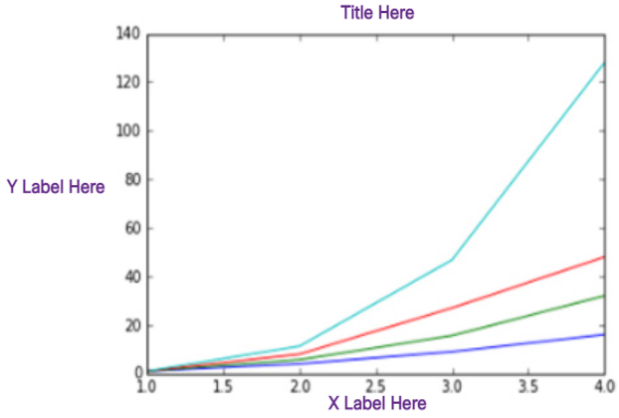


Рис. 10.2. Зміна стилів Pyplot за замовчуванням [1]

Якщо повторно використовувати один і той же вбудований код для декількох сюжетів, можна створювати спеціальний аркуш стилів. Посилаючись на користувальницький аркуш стилів, можна зробити так, щоб усі сюжети мали однакові функції стилю і уникнути помилок у вкладеному коді.

Щоб створити спеціальний аркуш стилів, потрібно відкрити текстовий редактор і додати рядок для кожного елемента в сюжеті, який потрібно налаштувати. Можна використовувати термінал Linux для створення та редагування текстового файлу [2].

Код на рис. 10.3 змінює деякі з багатьох налаштувань атрибутів сюжету.

```
root @ ~ / myfiles # cat mystyle.mplstyle
 axes.titlesize: 24
 axes.labelsize: 16
 lines.linewidth: 5
 xtick.labelsize: 10
 ytick.labelsize: 10
```

Рис. 10.3. Приклад користувацького стилю [1]

Збережіть таблицю стилів у відповідному місці за допомогою розширення .mplstyle. Ви можете зберігати його в будь-якому місці, але тоді потрібно буде надати інформацію про шлях, коли ви посилаєтесь на таблицю стилів. Наприклад, якщо ви зберігаєте таблицю стилів у "myfiles", то ваш код для посилання на таблицю стилів такий:

plt.style.use ('/ home / pi / notebooks / myfiles / mystyle.mplstyle')

Однак якщо зберігати конфігураційний файл у каталозі конфігурації matplotlib, інформація про шлях не потрібна. Використовуйте команду **get_configdir ()**, щоб знайти за замовчуванням файли конфігурації **matplotlib**. Потім скопіюйте таблицю стилів у це місце, як показано на рис. 10.4.

```
!pwd; ls
import matplotlib as mpl
mpl.get_configdir()
```

```
/home/pi/notebooks/myfiles
Custom Style Sheet Test.ipynb mystyle.mplstyle
My First Notebook.ipynb       temp-plot.html

u'/root/.config/matplotlib'
```

```
!cp mystyle.mplstyle /root/.config/matplotlib/mystyle.mplstyle
!ls /root/.config/matplotlib
```

```
mystyle.mplstyle
```

Рис. 10.4. Копіювання таблиці стилів до директорії [3]

Тепер потрібна лише назва аркуша стилів, щоб застосувати його до сюжету, як показано на рис. 10.5.

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.style.use('mystyle.mplstyle')
plt.plot([1,2,3,4], [1, 4, 9, 16])
plt.plot([1,2,3,4], [1, 5.7, 15.6, 32])
plt.plot([1,2,3,4], [1, 8, 27, 48])
plt.plot([1,2,3,4], [1, 11.3, 46.8, 128])
plt.xlabel('X Label Here')
plt.ylabel('Y Label Here')
plt.title('Title Here')
plt.show()
```

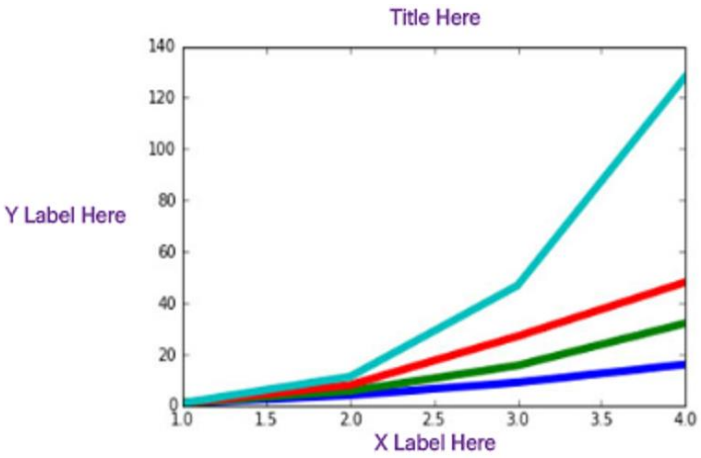


Рис. 10.5. Посилання на таблицю користувацького стилю [3]

У Matplotlib є кілька стилів, які можна викликати у коді. Використовуйте **print (plt.style.available)**, щоб переглянути стилі **matplotlib**, як показано на рис. 10.6.

```
print(plt.style.available)
```

```
[u'seaborn-darkgrid', u'seaborn-notebook', u'classic', u'seaborn-ticks',
 u'grayscale', u'bmh', u'seaborn-talk', u'dark_background', u'ggplot',
 u'fivethirtyeight', u'seaborn- colorblind', u'seaborn-deep',
 u'seaborn-whitegrid', u'seaborn-bright', u'seaborn- poster',
 u'seaborn-muted', u'seaborn-paper', u'seaborn-white', u'seaborn-pastel',
 u'seaborn-dark', u'seaborn-dark-palette']
```

Рис. 10.6. Доступні стилі в Matplotlib [3]

Matplotlib за замовчуванням використовує класичний стиль. Використовуйте також plt.style.use для зміни стилю. Наприклад, **plt.style.use ('grayscale')** змінить стиль на шкалу сірого, як показано на рис.10.7.

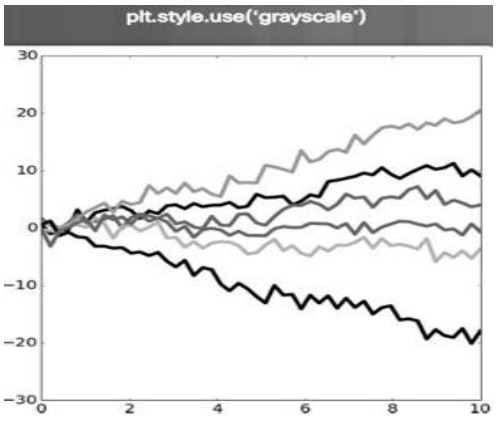


Рис. 10.7. Стиль «відтінки сірого» в Matplotlib [3]

На рис.10.8 показано інші приклади таблиць стилів, доступні у matplotlib.

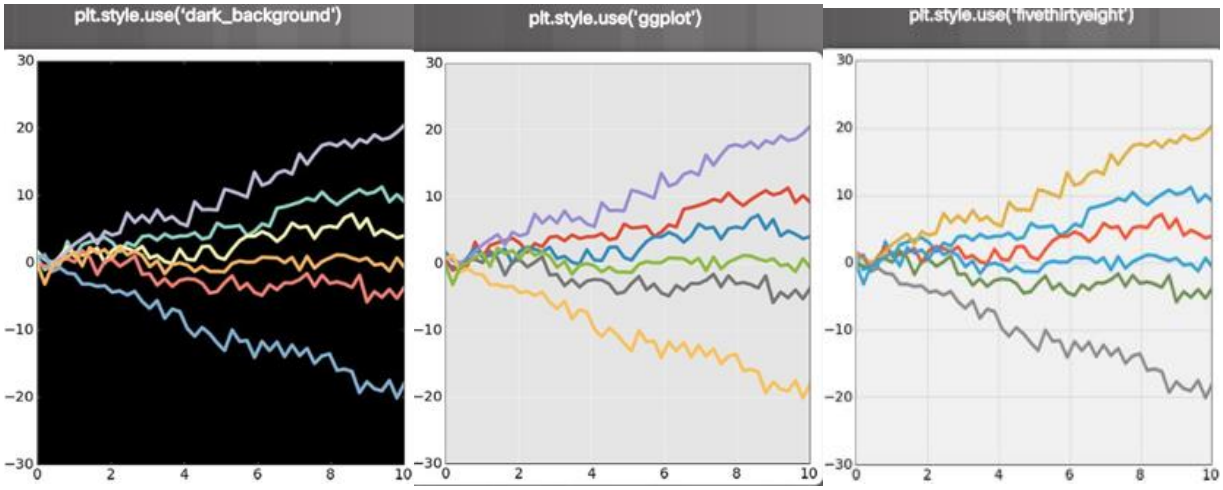


Рис. 10.8. Деякі інші стилі в Matplotlib [3]

10.2. Інструмент Plotly

Plotly – це потужний онлайн-інструмент, за допомогою якого можна швидко генерувати красиві візуалізації даних. Plotly має різноманітні ресурси для аналітиків даних та веб-розробників, включаючи бібліотеки API, перетворювачі фігур, додатки для Google Chrome та бібліотеку JavaScript з відкритим кодом.

На веб-сайті Plotly є велика кількість матеріалів, які доступні безкоштовно, включаючи графічний інтерфейс для створення та візуалізації даних. Щоб зберегти свої візуалізації, потрібно зареєструватися у вільному обліковому записі користувача, як показано на рис. 10.9.

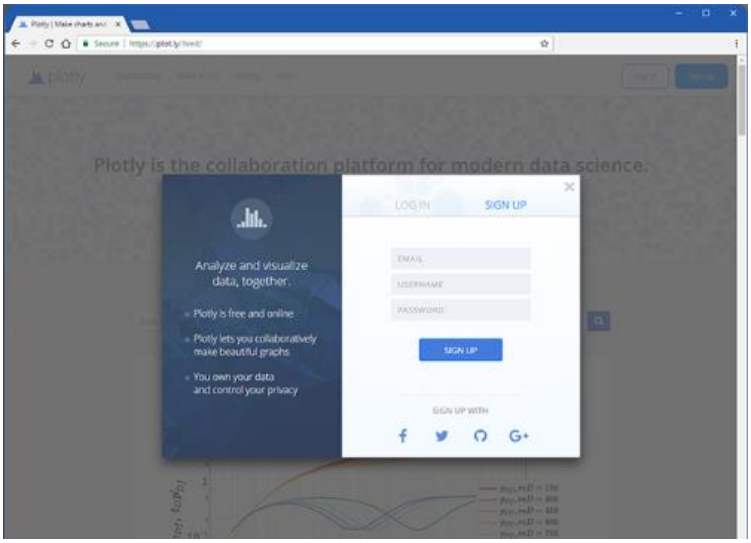


Рис. 10.9. Реєстрація у вільному обліковому записі користувача Plotly [4]

Після реєстрації ви можете переглядати веб-сайт з прикладами візуалізації публічних даних, здійснених іншими користувачами. Бібліотека Python plotly (plotly.py) – це інтерактивна бібліотека графіків із відкритим кодом, яка підтримує понад 40 унікальних типів діаграм, що охоплюють широкий спектр статистичних, фінансових, географічних, наукових та тривимірних випадків використання. Побудований поверх бібліотеки Plotly JavaScript (plotly.js), plotly.py дозволяє користувачам Python створювати прекрасні інтерактивні веб-візуалізації, які можуть відображатися в Jupyter Notebook, зберігатись у самостійних HTML-файлах або слугувати частиною Python- побудованих веб-програм за допомогою Dash [5].

Після успішного входу ви можете використовувати графічний графічний інтерфейс Plotly для створення кількох різних програмних продуктів. Як показано на рис. 10.10, натисніть Створити, потім – Діаграма, щоб почати створювати нову діаграму з нуля.

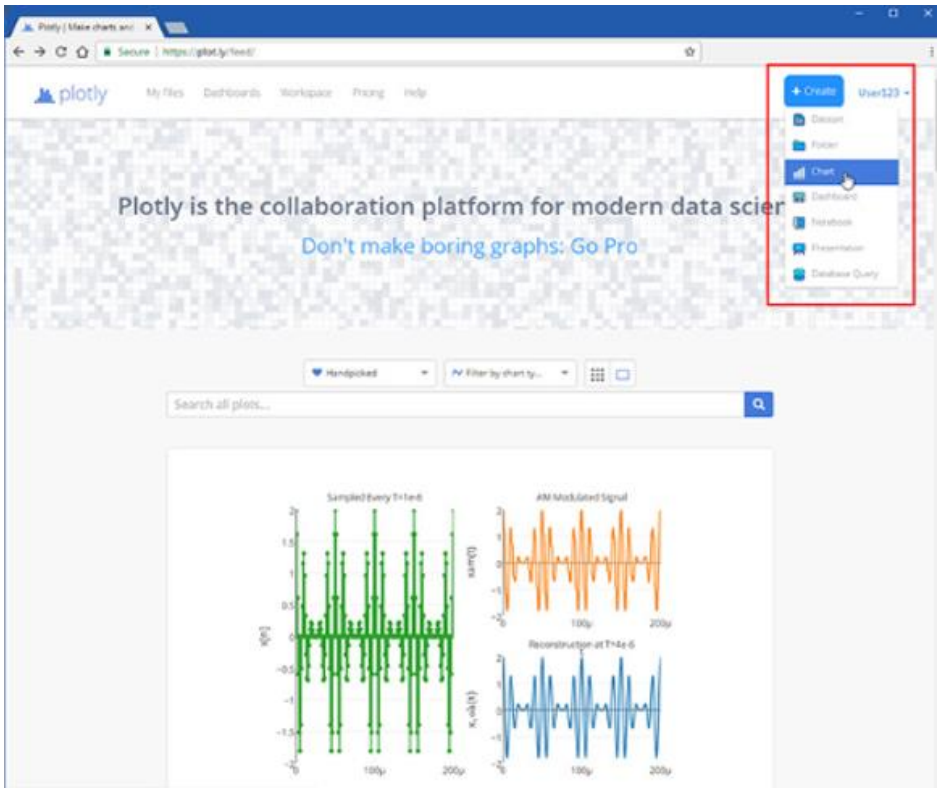


Рис. 10.10. Реєстрація у вільному обліковому записі користувача Plotly [4]

Клацніть «Тип діаграми», як показано на рис. 10.11. Помітьте всі доступні типи діаграм. Багато з них вільно використовуються. Інші потребують оновлення. Виберіть «Діаграма ліній» та введіть деякі дані тесту в електронну таблицю та інструменти для створення своєї ділянки. Експорт вимкнено, потрібно натиснути кнопку «Зберегти».

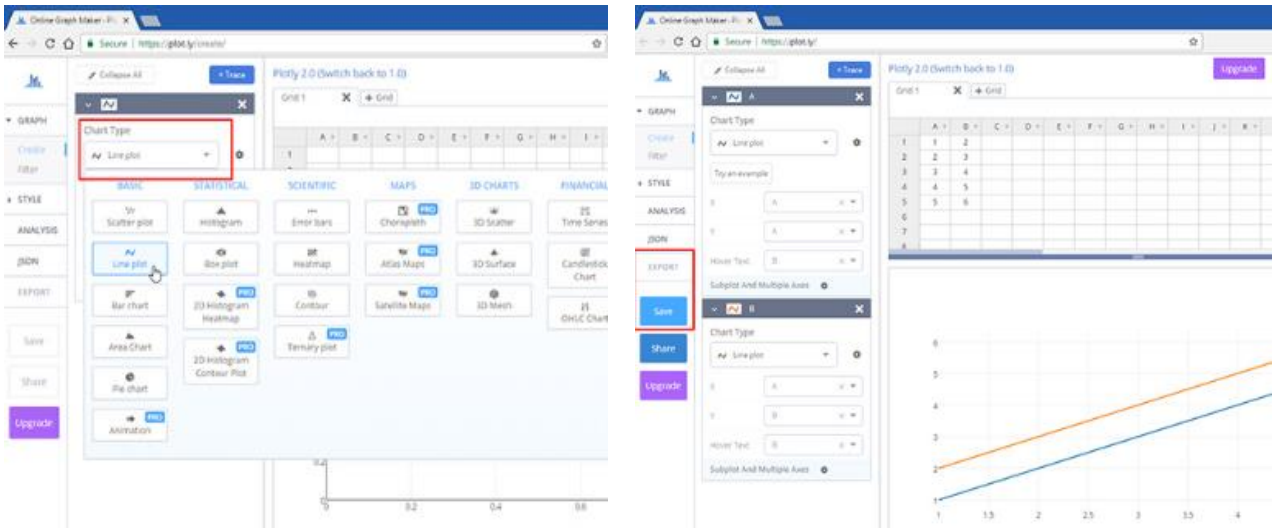


Рис. 10.11. Вибір типу діаграми та її збереження в Plotly [4]

Клацніть «Експорт», щоб переглянути доступні параметри зі сторінки «Створити». Ви обмежені кількома форматами файлів зображень та експортом HTML (рис. 10.12). Експорт HTML за замовчуванням у JavaScript.

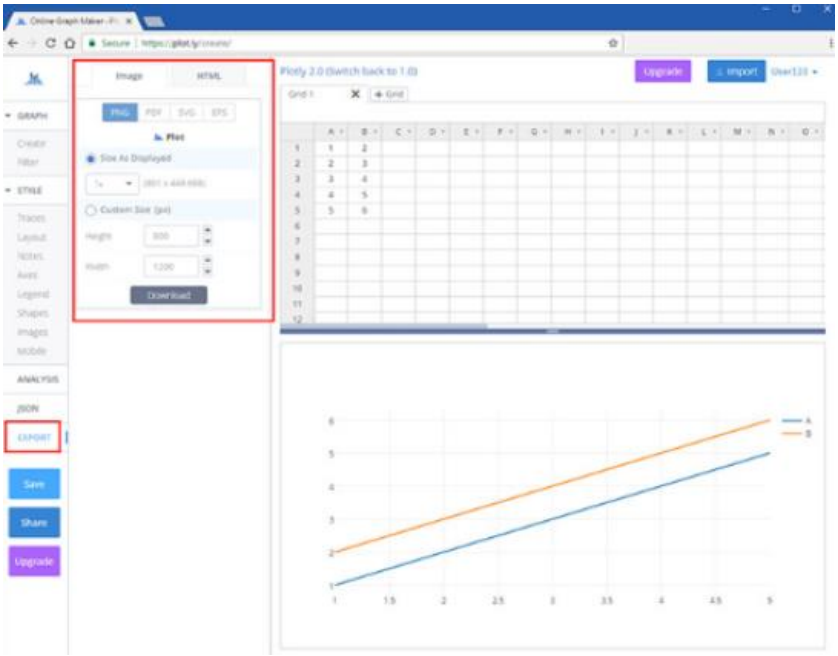


Рис. 10.12. Експорт діаграми в Plotly [4]

Щоб отримати доступ до всіх доступних форматів експорту, включаючи Python, натисніть своє ім’я користувача у верхньому правому куті, потім «Мої файли», як показано на рис. 10.13.

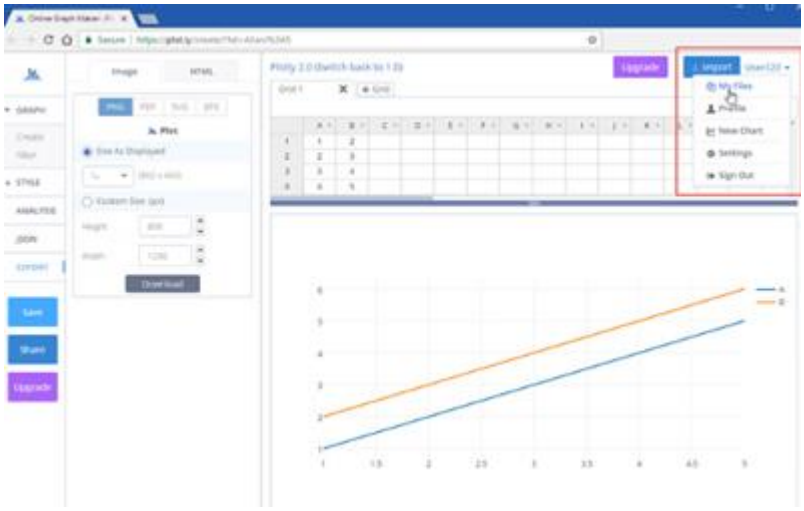


Рис. 10.13. Доступ до всіх доступних форматів експорту в Plotly [4]

У списку своїх файлів наведіть курсор миші на файл, який потрібно експортувати, і натисніть кнопку «Перегляд», як показано на рис. 10.14. Це перенесе вас на сторінку, де ви можете переглянути свій графік, дані, код та джерела. Вкладка «Код» дозволяє переглядати код на різних мовах. Ви можете скопіювати код і вставити його в текстовий файл.

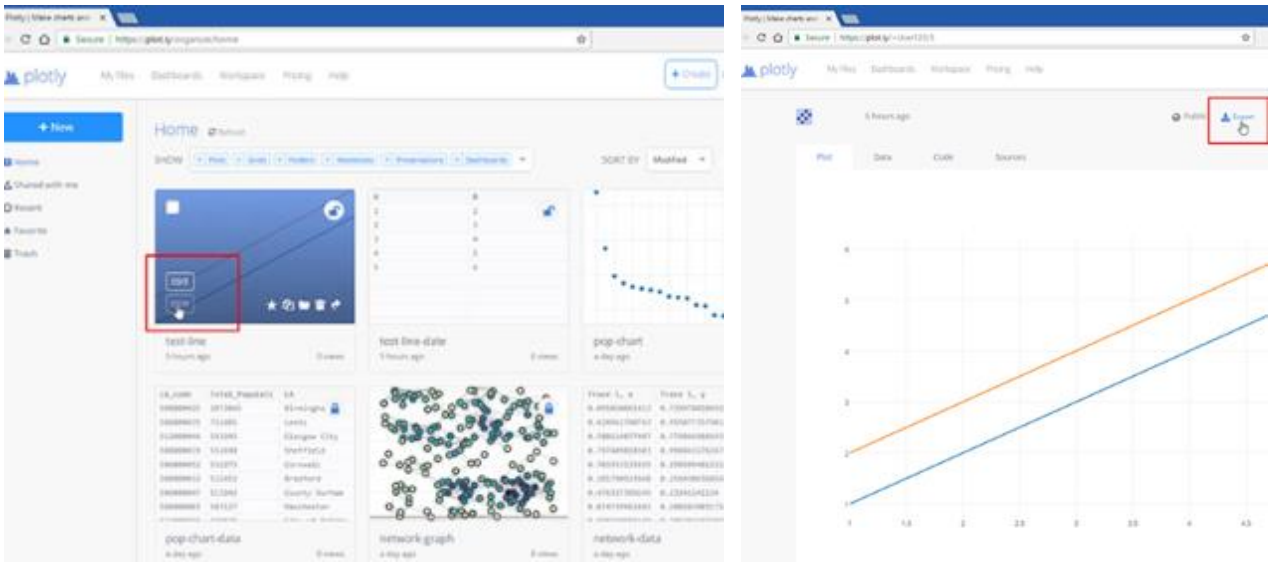


Рис. 10.14. Перегляд та експорт файлу в Plotly [4]

Ви також можете натиснути «Експорт», після цього відобразяться усі варіанти експорту, доступні на веб-сайті Plotly. Формати експорту коду показані на рис.10.15. Клацніть Python, щоб завантажити файл .py із усім кодом, необхідним для відтворення діаграми.

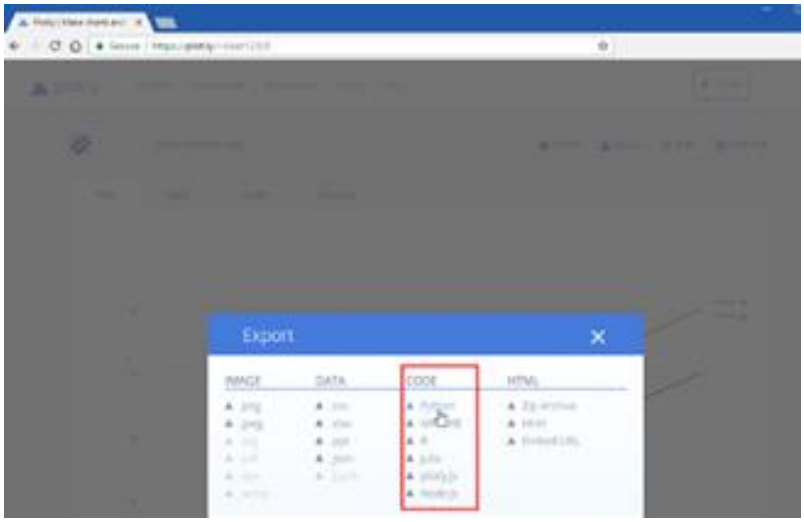


Рис. 10.15. Завантаження файлу в Plotly [4]

Також можна використовувати можливості Plotly для створення веб-сторінок в автономному режимі. Для цього потрібно встановити Plotly в Jupyter Notebook із таким кодом:

!pip install plotly

Код на рис.10.16 створить графік рядків і збереже його у поточному каталозі. Потім ви можете опублікувати веб-сторінку на своєму веб-сервері. Plotly має багато навчальних посібників та довідкових сторінок для даної платформи візуалізації даних (рис. 10.17).

```
import plotly
                from plotly.graph_objs import Scatter, Layout
                plotly.offline.plot({
                      "data": [Scatter(x=[1, 2, 3, 4], y=[4, 3, 2, 1])],
                      "layout": Layout(title="Hello World!")
                })
'file:///home/pi/notebooks/myfiles/temp-plot.html'
```

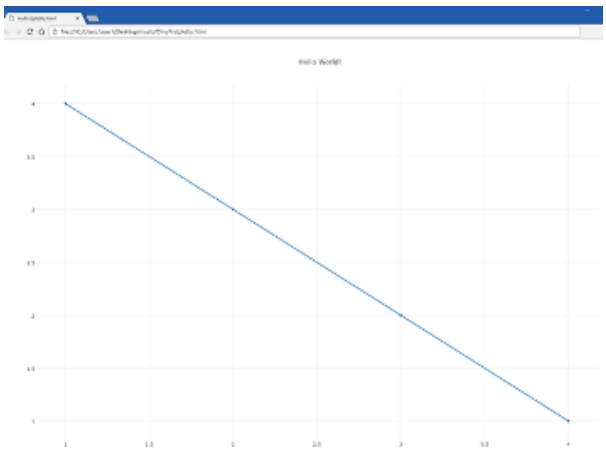


Рис. 10.16. Створення графіка та його збереження у поточному каталозі [4]

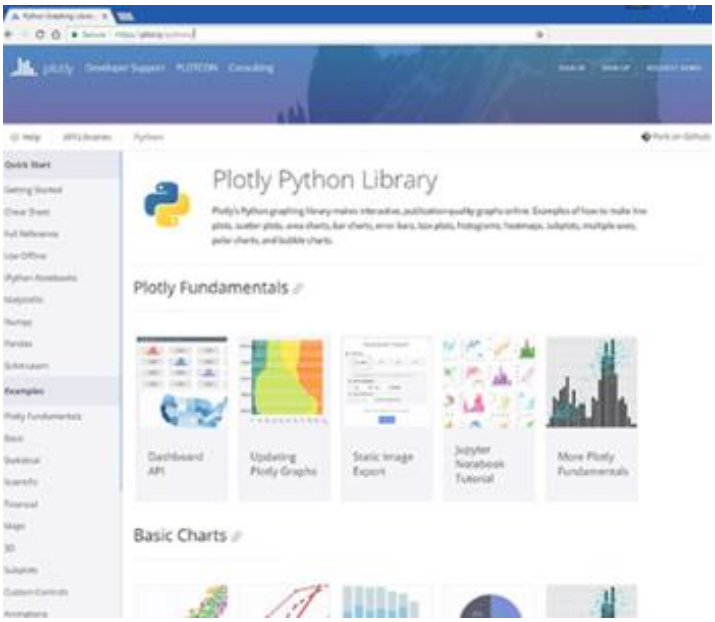


Рис. 10.17. Довідкові матеріали в Plotly [4]

10.3. Типи візуалізації даних

Визначення найкращого типу діаграми зазвичай залежить від відповідей на наступні питання:

- Скільки змінних ви збираєтеся показати?
- Скільки точок даних у кожній змінній?
- Чи ваші дані з часом чи ви порівнюєте предмети?

Лінійні діаграми є одним з найбільш часто використовуваних типів порівняльних діаграм, вони використовуються, коли у вас є неперервний набір даних, кількість точок даних є високою, і ви хочете показати тенденцію даних у часі.

Рис. 10.18. Лінійна діаграма [3]

Приклади:

- щоквартальні продажі за останні п’ять років;
- кількість клієнтів на тиждень у перший рік нового магазину роздрібної торгівлі.

Розглянемо деякі найкращі практики для побудови лінійних діаграм.

- Позначте свої осі.
- Час графіку на осі x (горизонтальний) та значення даних по осі y (вертикальна). Позначте обидві осі.
- Використовуйте суцільну лінію для даних, щоб підкреслити неперервність даних.
- Зведіть до мінімуму кількість наборів даних. У вас повинна бути вагома причина для побудови більш ніж чотирьох рядків. Бажно додати легенду до діаграми, щоб допомогти аудиторії зрозуміти, що вони переглядають.

Стовчикові діаграми розміщуються вертикально, вони, мабуть, є найпоширенішим типом діаграми, який використовується, коли потрібно відобразити значення конкретної точки даних та порівняти це значення у подібних категоріях.

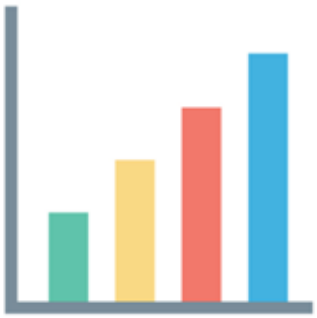


Рис. 10.19. Стовпчикова діаграма [3]

Приклади:

- населення країн;
- продажі для чотирьох найкращих автомобільних компаній;
- середні бали тесту для учнів шести класів з математики.

Розглянемо деякі найкращі практики для побудови стовпчикових діаграм.

- Позначте свої осі.
- Якщо час є одним із вимірів, його слід побудувати на осі x.
- Якщо час не є частиною даних, подумайте про те, щоб замовити висоту стовпчика для підйому чи спуску.
- Заповніть стовпчики суцільним кольором. Якщо ви хочете виділити один стовпець, подумайте про використання акцентного кольору та зробіть усі інші стовпці однаковими.
- Стовпці найкращі, коли на горизонтальній осі не більше семи категорій. Це допоможе глядачеві чітко бачити значення для кожного стовпця.
- Почніть значення осі y з нуля, щоб точно відобразити повне значення стовпця.
- Відстань між стовпцями має бути приблизно половиною ширини стовпця.

Гістограми схожі на стовпчикові діаграми, за винятком того, що вони розташовані горизонтально. Довші смужки означають більшу кількість. Їх краще використовувати, коли імена для кожної точки даних довгі.

Рис. 10.20. Гістограма [3]

Приклади:

- валовий внутрішній продукт (ВВП) 25 кращих країн;
- кількість автомобілів, проданих кожним торговим представником;
- оцінки іспиту для кожного учня з математики.

Розглянемо деякі найкращі практики для побудови гістограм.

- Позначте свої осі.
- Подумайте про впорядкування брусків так, щоб довжини переходили від найдовшої до найкоротшої. Тип даних, швидше за все, визначатиме, чи має бути найдовша смуга знизу чи вгорі.
- Залийте бруски суцільним кольором. Якщо ви хочете виділити одну смужку, подумайте про використання акцентного кольору та зробіть усі інші стовпці однаковими.
- Почніть значення осі x з нуля, щоб точно відобразити повне значення планки.
- Відстань між брусками повинна бути приблизно половиною ширини бруска.

Кругові діаграми використовуються для відображення складу статичного числа. Сегменти представляють відсоток від цієї кількості. Загальна сума сегментів повинна дорівнювати 100% (рис. 10.21).

Рис. 10.21. Кругова діаграма [3]

Приклади:

- щорічні витрати на корпорацію (наприклад, орендна плата, адміністративні послуги, комунальні послуги, виробництво);
- джерела енергії в країні (нафта, вугілля, газ, сонячна енергія, вітер тощо);
- результати опитування улюбленого типу фільму (наприклад, екшн, комедія, драма, наукова фантастика).

Розглянемо деякі найкращі практики для кругових діаграм.

- Зведіть кількість категорій до мінімуму, щоб глядач міг розмежувати сегменти. Після десяти і більше сегментів скибочки починають втрачати сенс і вплив.
- Якщо необхідно, консолідуйте менші сегменти в один сегмент з міткою, наприклад "Інше" або "Різне".
- Використовуйте різну кольорову або сіру шкалу для кожного сегмента.
- Упорядкуйте сегменти відповідно до розміру.
- Переконайтесь, що значення всіх сегментів дорівнює 100%.

Діаграми розсіювання використовують для кореляційних візуалізацій, для демонстрації розподілу великої кількості точок даних, для демонстрації кластеризації або ідентифікації залишків у даних тощо.

Рис. 10.22. Діаграма розсіювання [3]

Приклади:

- порівняння тривалості життя кожної країни з її ВВП;
- порівняння щоденних продаж морозива із середньою зовнішньою температурою;
- порівняння ваги з ростом кожної людини.

Розглянемо деякі найкращі практики побудови діаграм розсіювання.

- Позначте свої осі.
- Переконайтеся, що набір даних є достатньо великим, щоб забезпечити візуалізацію для кластеризації.
- Почніть значення осі y з нуля, щоб точно відобразити повне значення планки. Значення осі x залежатиме від даних. Наприклад, вікові діапазони можуть бути позначені на осі x.
- Якщо графік розсіювання показує кореляцію між осі x і y, розгляньте можливість додавання лінії тренду.
- Не використовуйте більше двох ліній тренду.

Коли потрібно додати третю змінну до діаграми розкидання, подумайте про використання бульбашкової діаграми. Наприклад, графік розсіювання, що показує співвідношення між тривалістю життя та ВВП, може бути покращений за рахунок збільшення розміру кожної точки даних для представлення населення.

10.4. Візуалізація аномалій

У якості допоміжного засобу для візуалізації тривимірної графіки може бути корисно переглядати дані з різних кутів. Для цього потрібно імпортувати

інтерактивний клас із бібліотеки ipywidgets, щоб додати інструменти для регулювання азимуту та висоти 3D-графіку. Азимут дозволить обертати ділянку по горизонталі. Підняття дозволить повертати ділянку вертикально (рис. 10.23).

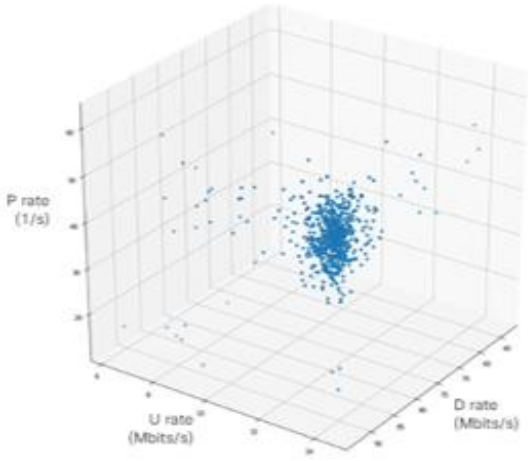


Рис. 10.23. Зміна азимуту та висоти 3D-сюжету [3]

Аномалії можуть представляти аномальні дані або значення. Дані можуть бути пошкоджені або спотворені багатьма факторами під час вимірювання, передачі чи зберігання. Ці значення настільки далеко відхиляються від очікуваних значень, що можуть спотворити результати аналізу. Ці спостереження часто знімаються з набору даних після ретельного розгляду.

Є й інші типи аномалій, які можуть представляти серйозні проблеми з предметом, який вимірюється. Наприклад, незвичайно високі температури або вібраційні вимірювання, зроблені датчиками, приєднаними до машини, можуть свідчити про те, що частина з них готова вийти з ладу. У цьому випадку додаток для аналізу потокових даних на IoT може надіслати сигнал тривоги, який би попередив обслуговуючий персонал про те, що машина потребує уваги.

На рис. 10.23 кілька точок даних лежать поза межами кластерної області. Це – аномалії. Аномалії можна ідентифікувати, виявивши точки, що лежать понад середнього. Виміряючи різницю між координатами x, y та z кожної точки даних та середніми координатами x, y та z, ми отримуємо список відстаней для кожної точки даних.

Щоб виявити аномалії, потрібно визначити межу прийняття рішення, яка визначає, чи нормальна точка даних або аномалія. Для цього спочатку потрібно нормалізувати дані про відстань, встановивши найбільшу відстань 1. Потім визначити поріг між 0 і 1 для межі рішення.

9.5. Використання бібліотек Folium та Leaflet.js для побудови карт

Бібліотека для побудови карт Folium поєднує силу Python із можливостями відображення бібліотеки Leaflet.js. Folium дозволяє взяти дата фрейми Python та відобразити їх на інтерактивній карті Leaflet (рис. 10.24).

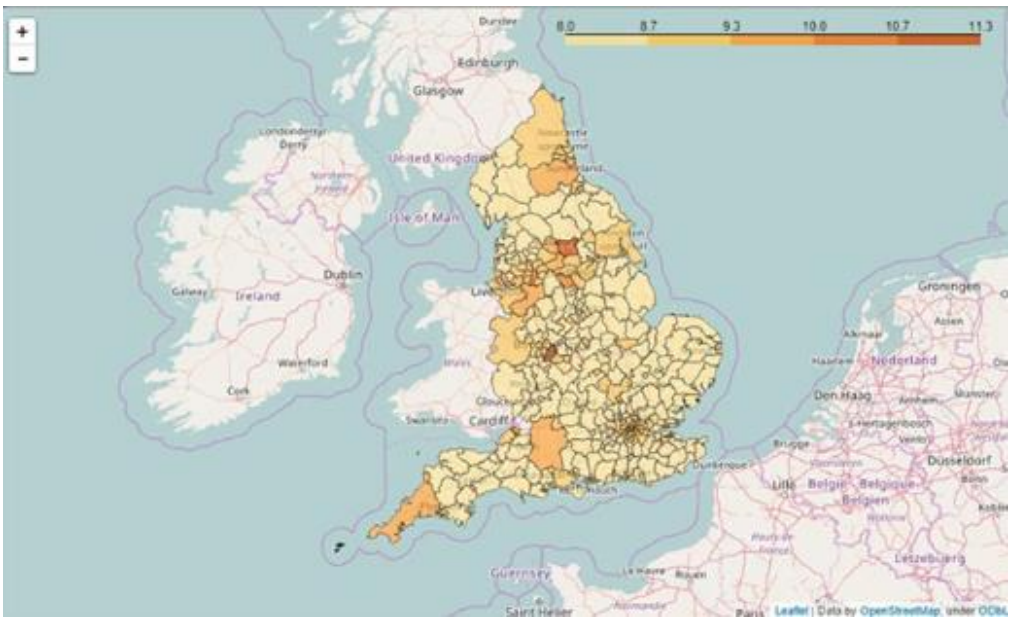


Рис. 10.24. Приклад використання Folium для відображення даних на карті Leaflet [3]

Folium Tilesets – це це набір растрових або векторних даних, які можуть відображати карту на мобільних пристроях або в браузері. Бібліотека Folium підтримує ряд різних tileset, включаючи OpenStreetMap, Mapbox і Stamen. За замовчуванням Folium використовує tileset OpenStreetMap. Карти Mapbox і Stamen можна вказати за допомогою атрибута tiles (Mapbox потребує облікового запису користувача, щоб отримати маркери доступу API). Імпортувавши бібліотеку Folium та використовуючи tileset OpenStreetMap, можна створити карту, вказавши набір координат (рис. 10.25).

```
import folium
svmap = folium.Map(location=[37.3861, -122.0839])
svmap
```

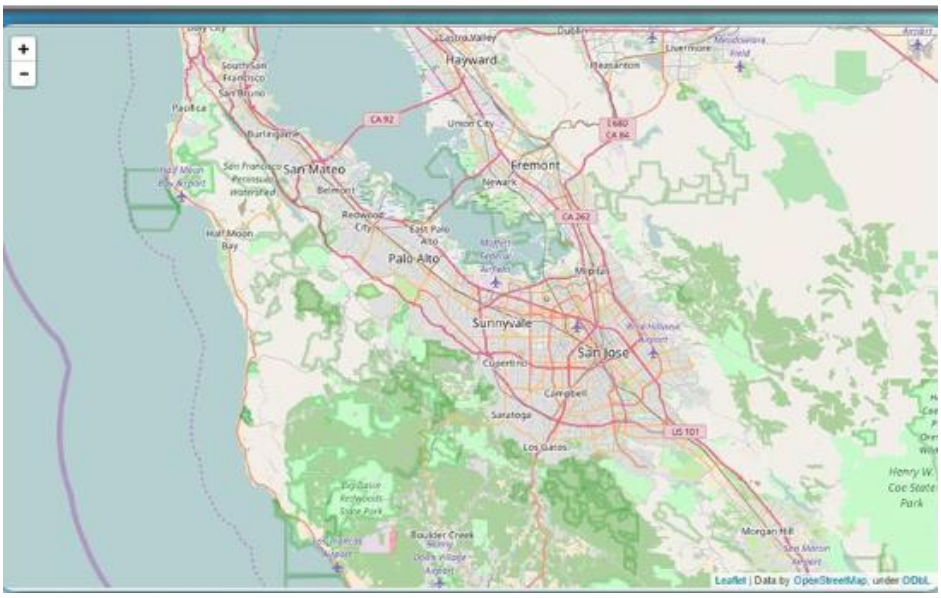


Рис. 10.25. Приклад використання бібліотеки Folium [3]

Висновок до лекції 10

Pyplot – це розширення matplotlib, що включає набір функцій стилю для створення та налаштування графіків, дозволяє повторно використовувати один і той же вбудований код для кількох графіків та створювати спеціальні таблиці стилів. Plotly – це потужний онлайн-інструмент для швидкого створення візуалізацій даних. В аналізі даних лінійні діаграми використовуються для неперервних наборів даних, де кількість точок даних велика, і потрібно показати тенденцію у часі. Стовпчикові діаграми розташовані вертикально. Цей тип діаграми використовується, коли потрібно відобразити значення конкретної точки даних і порівняти це значення за подібними категоріями. Гістограми подібні до стовпчикових діаграм, за винятком того, що вони розташовані горизонтально. Кругові діаграми використовуються для відображення складу статичного числа. Сегменти представляють відсоток від цього числа. Загальна сума сегментів дорівнює 100%. Графіки розсіювання використовуються для візуалізації кореляції або розподілу великої кількості точок даних та кластеризації або виявлення відхилень у даних. Для візуалізації даних на інтерактивній карті у Python використовується бібліотека Folium.

Питання для закріплення

1. Для чого використовується модуль Pyplot?
2. Опишіть інструмент Plotly.
3. Які типи візуалізації даних ви знаєте?
4. Що таке аномалії даних?
5. Для чого використовується бібліотека Folium у Python?

Список рекомендованої літератури

1. Customizing Matplotlib with style sheets and rcParams // Електронний ресурс. Режим доступу: https://matplotlib.org/tutorials/introductory/customizing.html
2. How to Quickly Create a Text File Using the Command Line in Linux // Електронний ресурс. Режим доступу: https://www.howtogeek.com/199687/how-to-quickly-create-a-text-file-using-the-command-line-in-linux/
3. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
4. Plotly Python Open Source Graphing Library // Електронний ресурс. Режим доступу: https://plot.ly/python/
5. Getting Started with Plotly in Python // Електронний ресурс. Режим доступу: https://plot.ly/python/getting-started/

Лекція 11.
Аналіз даних в R.
Фактори, списки, фрейми та дії над ними

План лекції

11.1. Історія розвитку мови R.
11.2. Можливості мови R.
11.3. Об'єкти, пакети, функції.
11.4. Вектори, матриці та операції над ними в R.
11.5. Фактори, списки, фрейми та дії над ними.

11.1. Історія розвитку мови R

Уперше мову програмування R було впроваджено на початку 1990-х років Робертом Джентльменом та Россом Іхакою, викладачами Оклендського університету (рис. 11.1). Перша літера їх імен і дала назву новій системі.

Рис. 11.1. Розробники мови R (Robert Gentleman, Ross Ihaka)

Мова R була створена як аналог мови для статистичних обчислень S, яка була реалізована Джоном Чемберсом, Ріком Беккером, Тревором Хасті, Алланом Уілксом та іншими в Bell Labs в середині 1970-х років та опублікована на початку 1980-х. Роберт і Росс створили R як проект з відкритим кодом у 1995 році. З 1997 року проєктом R керує R Core Group.

В лютому 2000 року вийшов перший реліз R. На сьогодні створюються нові інструменти для R, існує понад 10 000 створених користувачами бібліотек для покращення функціональності R. Ці пакети мають краудсорсинг для перевірки якості та підтримки визнаних лідерів у кожній галузі [1].

11.2. Можливості мови R

R – це мова статистичного програмування, яку спеціалісти з обробки даних використовують у всьому світі – від картографування широких соціальних та маркетингових тенденцій в Інтернеті.

Спеціаліст з оброблення великих даних може використовувати два чудові інструменти: R і Python, проте навчання статистичному моделюванню є важливішим, ніж вивчити мову програмування. Мова програмування – це лише інструмент для аналізу даних. Найбільш важливим завданням в науці про дані є те, як спеціаліст працює з даними, важливими є імпорт, очищення, підготовка, розробка функцій, вибір функцій. Це повинно бути основним фокусом. Якщо для аналізу даних намагатися вивчати R і Python одночасно, не маючи достатніх знань в області статистики, це нерозумно. Робота аналітика полягає в тому, щоб розуміти дані, маніпулювати ними і розкривати найкращий підхід.

Основна аудиторія науки про дані – професіонали бізнесу. Кілька років тому мова R була не такою структурованою, як інші інструменти програмування. Щоб подолати цю серйозну проблему, Хедлі Уікхем створив колекцію пакетів під назвою tidyverse. Правила гри змінилися в кращу сторону. Маніпулювання даними стає тривіальним і інтуїтивно зрозумілим процесом. Створення графіки також не є викликає труднощів. Кращі алгоритми для машинного навчання можуть бути реалізовані за допомогою R. Такі пакети, як Keras і TensorFlow,

дозволяють вирішувати задачі машинного навчання. Аналіз даних, такий як кластеризація, кореляція і редукція, також може виконуватися за допомогою R.

Мова R також має пакет для виконання Xgboost, одного з кращих алгоритмів для змагань Kaggle. R може «спілкуватися» з іншими мовами програмування. Можна назвати Python, Java, C ++ в R. Аналітик можете зв'язати R з різними базами даних, такими як Spark або Hadoop. Мова R дозволяє розпаралелювати операції, щоб прискорити обчислення. Пакет для паралельних обчислень дозволяє виконувати завдання в різних ядрах машини.

11.3. Об'єкти, пакети, функції

Мова R належить до високорівневих об'єктно-орієнтованих мов програмування. Для неспеціаліста суворе визначення поняття "об'єкт" є досить абстрактним. Однак для простоти можна називати об'єктами все, що було створено в процесі роботи з R. В R виділяють два основних типи об'єктів:

1. Об'єкти, призначені для зберігання даних ("data objects") – це окремі змінні, вектори, матриці та масиви, списки, фактори, таблиці даних.

2. Функції ("function objects") – це зазначені програми, призначені для створення нових об'єктів або виконання певних дій над ними.

Об'єкти середовища R, призначені для колективного і вільного використання, комплектуються в пакети, що об'єднуються подібною тематикою або методами обробки даних. Є певна відмінність між термінами пакет ("package") і бібліотека ("library"). Термін "library" визначає директорію, яка може містити один або кілька пакетів. Термін "package" позначає сукупність функцій, HTML-сторінок, посібників і прикладів об'єктів даних, призначених для тестування або навчання. Пакети встановлюються в певній директорії операційної системи або, в невстановленому вигляді, можуть зберігатися і поширюватися в архівних * .zip файлах Windows (версія пакету має кореспондуватися з конкретною версією R). Усі пакети R відносяться до однієї з трьох категорій: базові ("base"), рекомендовані ("recommended") та інші, встановлені користувачем. Отримати їх список на конкретному комп'ютері можна, подавши команду library () або:

```
installed.packages (priority = "base")
installed.packages (priority = "recommended")
# Отримання повного списку пакетів
packlist <- rownames (installed.packages ())
# Виведення інформації в буфер обміну в форматі для Excel
write.table (packlist, "clipboard", sep = "\ t", col.names = NA)
```

Базові та рекомендовані пакети зазвичай включаються в інсталяційний файл R. Зрозуміло, немає необхідності відразу встановлювати "про запас" багато різних пакетів. Для установки пакета досить в командному вікні R Console вибрати пункт меню "Пакети> Встановити пакет (и)" або ввести, наприклад, команду:

```
install.packages (c ( "vegan", "xlsReadWrite", "car"))
```

Таблиця 11.1. Деякі базові пакети R

Пакети R	**Призначення**
base	Базові конструкції R
compiler	Компілятор пакетів R
datasets	Набір таблиць з даними для тестування і демонстрації функцій
graphics	Базові графічні функції
grDevices	Драйвери графічних пристроїв, палітри кольорів, шрифти
grid	Функції створення графічних шарів
methods	Компоненти об’єктно-орієнтованого програмування (класи, методи)
splines	Функції роботи з регресійний сплайнами різного типу
stats	Базові функції статистичного аналізу
stats4	Методи статистичних функцій класу S4
tcltk	Компоненти інтерфейсу з користувачем (меню, бокси вибору та ін.)
tools	Інформаційна підтримка, адміністрування та документування
utils	Різні утиліти налагодження, вводу-виводу, архівування тощо

Усі об'єкти даних (а, отже, і змінні) в R можна розділити на такі класи (тобто типи об'єктів):

° numeric – об'єкти, до яких відносяться цілочисельні (integer) і дійсні числа (double);

° logical – логічні об'єкти, які приймають тільки два значення: FALSE (F) і TRUE (T);

° character – символьні об'єкти (значення змінних задаються в подвійних, або одинарних лапках).

В R можна створювати імена для різних об'єктів (функцій або змінних) як на латиниці, так і на кирилиці, але слід врахувати, що а (кирилиця) і a (латиниця) – це два різних об'єкта. Крім того, R чутлива до регістру, тобто рядкові і заголовні букви в ній відрізняються. Імена змінних (ідентифікатори) в R повинні починатися з літери (або крапки .) і складатися з букв, цифр, знаків крапки і підкреслення.

За допомогою команди ?<Ім'я> можна перевірити, чи існує змінна або функція із зазначеними ім'ям. Перевірка на приналежність змінної до певного класу перевіряється функціями is.numeric (<ім'я_об'єкта>), is.integer (<ім'я>), is.logical (<ім'я>), is.character (<ім'я>), а для перетворення об'єкта в інший тип можна використовувати функції as.numeric (<ім'я>), as.integer (<ім'я>), as.logical (<ім'я>), as.character (<ім'я>).

В R існує ряд спеціальних об'єктів:

° Inf – позитивна чи негативна нескінченність (зазвичай результат ділення дійсного числа на 0);

° NA – "відсутнє значення" (Not Available);

° NaN – «не число" (Not a Number).

Перевірити, чи стосується змінна до якого-небудь з цих спеціальних типів, можна, відповідно, функціями is.finite (<ім'я>), is.na (<ім'я>) та is.nan (<ім'я>).

Вираз (expression) мови R являє собою поєднання таких елементів, як оператор присвоювання, арифметичні або логічні оператори, імена об'єктів та імена функцій. Результат виконання виразу, як правило, відразу відображається

в командному або графічному вікні. Однак при виконанні операції присвоювання результат зберігається у відповідному об'єкті і на екран не виводиться. В якості оператора присвоєння в R можна використовувати або символ "=", або пару символів "<-" (присвоювання певного значення об'єкту зліва) або "->" (привласнення значення об'єкту праворуч). Хорошим стилем програмування вважається використання "<-".

Вирази мови R організовуються в скрипті по рядках. В одному рядку можна ввести кілька команд, розділяючи їх символом ";". Одну команду можна також розташувати на двох (і більше) рядках.

11.4. Вектори, матриці та операції над ними в R

Вектор в R – це іменований одновимірний об'єкт, що містить набір однотипних елементів (числові, логічні, або текстові значення – ніякі їх поєднання не допускаються). Для створення векторів невеликої довжини в R використовується функція конкатенації c() (від "concatenate" – об'єднувати, пов'язувати). В якості аргументів цієї функції через кому перераховують об'єднуються в вектор значення, наприклад:

```
my.vector <- c (1, 2, 3, 4, 5)
my.vector
[1] 1 2 3 4 5
```

Для створення векторів, що містять послідовну сукупність чисел, зручна функція seq () (від "sequence" - послідовність). Так, вектор з ім'ям S, що містить сукупність цілих чисел від 1 до 7, можна створити наступним чином:

```
S <- seq (1,7)
S
[1] 1 2 3 4 5 6 7
```

Ідентичний результат буде отримано за допомогою команди

```
S <- 1: 7
S
[1] 1 2 3 4 5 6 7
```

Як додатковий аргумент функції seq () можна задати крок збільшення чисел:

```
S <- seq (from = 1, to = 5, by = 0.5)
S
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Вектори, що містять однакові значення, створюють за допомогою функції rep() (від "repeat" – повторювати). Наприклад, для формування текстового вектора Text, який буде містити п'ять значень "test", слід виконати команду:

```
Text <- rep ( "test", 5)
Text
[1] "test" "test" "test" "test" "test"
```

Система R здатна виконувати найрізноманітніші операції над векторами. так, кілька векторів можна об'єднати в один, використовуючи вже розглянуту вище функцію конкатенації:

```
v1 <- c (1, 2, 3)
v2 <- c (4, 5, 6)
V <- c (v1, v2)
V
[1] 1 2 3 4 5 6
```

Якщо спробувати об'єднати, наприклад, текстовий вектор з числовим, повідомлення про помилково не з'явиться – програма просто перетворює всі значення в текстові. Для роботи с певним елементом вектора необхідно мати спосіб відрізняти його від інших елементів. Для цього при створенні вектора всіх його компонентів автоматично присвоюються індексні номери, починаючи з 1. Щоб звернутися до конкретного елементу, необхідно вказати ім'я вектора і індекс цього елемента в квадратних дужках:

```
# Створимо числовий вектор y, що містить 5 числових значень:
y <- c (5, 3, 2, 6, 1)
# Перевіримо, чому дорівнює третій елемент вектора y:
y [3]
[1] 2
```

Використовуючи індексні номери, можна виконувати різні операції з обраними елементами різних векторів. Індексування є потужним інструментом, що дозволяє створювати сукупності значень відповідно до визначених критеріїв. Наприклад, для виведення на екран 3-го, 4-го і 5-го значень вектора у необхідно виконати команду:

```
y [3: 5]
[1] 2 6 1
```

З цього ж вектора ми можемо вибрати, наприклад, тільки перше і четверте значення, використовуючи відому нам функцію конкатенації c():

```
y [c (1, 4)]
[1] 5  6
```

Схожим чином ми можемо видалити перше і четверте значення з вектора y, застосувавши знак "мінус" перед функцією конкатенації:

```
y [-c (1, 4)]
[1] 3 2 1
```

В якості критерію для вибору значень може служити логічний вираз. Для прикладу виберемо з вектора y всі значення, які більші 2:

```
y [y> 2]
[1] 5 3 6
```

Індексування є також зручним інструментом для внесення виправлень в наявних векторах. Наприклад, так можна виправити друге значення вектора z з 0.1 на 0.3:

```
z [2] <- 0.3
```

Для упорядкування значень вектора за зростанням або спаданням використовують функцію sort () в поєднанні з аргументом decreasing = FALSE або decreasing = TRUE відповідно:

```
sort (z) # за замовчуванням decreasing = FALSE
[1] 0.3 0.5 0.6
sort (z, decreasing = TRUE)
[1] 0.6 0.5 0.3
```

Матриця являє собою двовимірний вектор. В R для створення матриць служить однойменна функція:

```
my.mat <- matrix (seq (1, 16), nrow = 4, ncol = 4)
my.mat
     [, 1]  [2]  [3]  [4]
[1,]  1     5     9   13
[2,]  2     6    10   14
[3,]  3     7    11   15
[4,]  4     8    12   16
```

За замовчуванням заповнення матриці відбувається по стовпчиках, тобто перші чотири значення входять в перший стовпець, наступні чотири значення – у другий стовпець, і т.д. Такий порядок заповнення можна змінити, надавши спеціальному аргументу byrow (від "by row" – по рядках) значення TRUE:

```
my.mat <- matrix (seq (1, 16), nrow = 4, ncol = 4, byrow = TRUE)
my.mat
     [, 1]  [2]  [3]  [4]
[1,]  1     2     3    4
[2,]  5     6     7    8
[3,]  9    10    11   12
[4,] 13    14    15   16
```

В якості заголовків рядків і стовпців створюваної матриці автоматично виводяться відповідні індексні номери (рядки: [1,], [2,], і т.д .; стовпці: [, 1], [2], і т.д.). Для додання призначених для користувача заголовків рядках і стовпцях матриць використовують функції rownames() і colnames() відповідно. Наприклад, для позначення рядків матриці my.mat буквами A, B, C і D необхідно виконати наступне:

```
rownames (my.mat) <- c ( "A", "B", "C", "D")
my.mat
   [, 1]  [2]  [3]  [4]
A    1     2     3    4
```

B 5 6 7 8
C 9 10 11 12
D 13 14 15 16

У матриці my.mat є 16 значень, які вміщуються в наявні 4 рядки і 4 стовпці. Якщо спробувати вмістити вектор з 12 чисел в матрицю того ж розміру, R заповнить відсутні значення за рахунок "зациклення" короткого вектора:

```
my.mat2 <- matrix (seq (1, 12), nrow = 4, ncol = 4, byrow = TRUE)
my.mat2
     [, 1] [2] [3] [4]
[1,]  1    2    3    4
[2,]  5    6    7    8
[3,]  9   10   11  12
[4,]  1    2    3    4
```

Як бачимо, для заповнення комірок останнього рядка матриці my.mat2 програма знову використовувала числа 1, 2, 3, і 4.

Альтернативний спосіб створення матриць полягає в застосуванні функції dim () (від "dimension" – розмірність). Так, матрицю my.mat ми могли б сформувати з одновимірного вектора наступним чином:

```
my.mat <- 1:16
# Задаємо розмірність 4x4 вектору my.mat:
dim (my.mat) <- c (4, 4)
my.mat
   [, 1] [2] [3] [4]
[1,]  1   5    9   13
[2,]  2   6   10  14
[3,]  3   7   11  15
[4,]  4   8   12  16
```

Функція dim () дозволяє перевірити розмірність вже наявної матриці (або таблиці даних):

```
dim (my.mat)
[1] 4  4
```

Матрицю можна зібрати також з декількох векторів, використовуючи функції cbind () (від colum і bind – стовпець і пов'язувати) або rbind () (від row і bind – рядок і пов'язати):

```
# створимо чотири вектори однакової довжини:
a <- c (1, 2, 3, 4)
b <- c (5, 6, 7, 8)
d <- c (9, 10, 11, 12)
e <- c (13, 14, 15, 16)
# об'єднаємо вектори за допомогою функції cbind ():
cbind (a, b, d, e)
      a b d e
[1,] 1 5 9 13
[2,] 2 6 10 14
[3,] 3 7 11 15
[4,] 4 8 12 16
# Об'єднаємо ті ж вектори за допомогою функції rbind ():
rbind (a, b, d, e)
 [, 1] [2] [3] [4]
a   1    2   3   4
b   5    6   7   8
d   9   10 11 12
e   13 14 15 16
```

Практично усі векторні операції однаково застосовуватися щодо матриць і масивів. Так, за допомогою індексу ми можемо отримувати з матриць необхідні елементи і далі піддавати їх необхідним перетворенням.

Розглянемо кілька прикладів:

1. Витягти елемент матриці my.mat, розташований на перетині 2-го рядка і 3-го стовпця:

```
my.mat [2, 3]
[1] 7
```

2. Витягти з матриці всі елементи, що знаходяться в 4-м стовпці (для цього номера рядків перед коми можна не вказувати):

```
my.mat [, 4]
[1] 4 8 12 16
```

3. Витягти з матриці всі елементи, що знаходяться в 1-му рядку (у цьому випадку немає необхідності вказувати номери стовпців):

```
my.mat [1,]
[1] 1 2 3 4
```

4. Перемножимо 1-й і 4-й стовпці матриці (поелементно):

```
my.mat [, 1] * my.mat [, 4]
[1] 4 40 108 208
```

При необхідності матрицю можна транспонувати (міняти місцями рядки і стовпці) за допомогою функції t () (від transpose):

```
t (my.mat)
     A B C D
[1,] 1 5 9 13
[2,] 2 6 10 14
[3,] 3 7 11 15
[4,] 4 8 12 16
```

11.5. Фактори, списки, фрейми та дії над ними

У статистиці дані дуже часто групують відповідно до тих чи інших ознак, наприклад, статей, соціального становища, стадії хвороби, місцем відбору проб тощо. В R існує спеціальний клас векторів – фактори (factors), які призначені для зберігання кодів відповідних рівнів номінальних ознак. Часто рівні факторів кодують у вигляді чисел. У таких випадках дуже важливо "проінструктувати" програму так, щоб вона "розпізнавала" рівні номінальної змінної від чисел як таких. Припустимо, що в експерименті з випробування ефективності нового медичного препарату було задіяно 10 пацієнтів-добровольців, з яких шість пацієнтів приймали новий препарат, а четверо інших – плацебо (наприклад,

таблетки активованого вугілля). Для позначення учасників цих двох груп ми можемо використовувати коди 1 (препарат) і 0 (плацебо). Відповідно, інформацію про всіх десяти учасників експерименту ми могли б зберегти як наступного вектора:

```
treatment <- c (1, 1, 1, 1, 1, 1, 0, 0, 0, 0)
treatment
[1] 1 1 1 1 1 1 0 0 0 0
```

При такому підході, однак, програма буде "розглядати" вектор treatment в якості числового. Це буде помилкою з нашого боку, оскільки нуль і одиниця позначають лише два рівня номінальної змінної. З таким же успіхом ми могли б використовувати, наприклад, 10 для позначення контрольної групи пацієнтів (тобто пацієнтів приймали плацебо) і 110 для позначення пацієнтів, які приймали досліджуваний препарат. Для перетворення числового (або текстового) вектора в фактор в R існує однойменна функція factor ():

```
treatment <- factor (treatment, levels = c (0, 1))
treatment
[1] 1 1 1 1 1 1 0 0 0 0
Levels: 0 1
```

Зверніть увагу на те, що тепер при виведенні вмісту об'єкта treatment програма підказує нам, що цей об'єкт є фактором з двома рівнями (Levels: 0 1). Додатково переконатися в цьому можна за допомогою команди class (treatment):

```
class (treatment))
[1] "factor"
```

Більш надійним підходом, що дозволяє не заплутатися при виконанні аналізу, є кодування рівнів факторів за допомогою текстових значень, а не чисел. Наприклад, в нашому прикладі можна привласнити значення yes пацієнтам, які брали препарат, і значення no пацієнтам з контрольної групи. Ми можемо перекодувати рівні вже наявного фактора treatment за допомогою функції levels ():

```
levels (treatment) <- c ( "no", "yes")
```

treatment

[1] yes yes yes yes yes yes no no no no

Levels: no yes

Зауважте, що при виведенні вмісту вектора treatment коди пацієнтів не укладені в подвійні лапки, як це зазвичай буває у випадку з текстовими значеннями. Це є одним із зовнішніх ознак того, що ми маємо справу саме з фактором, а не з текстовим вектором, що містить шість значень "yes" і чотири значення "no". Ті ж фактори легко перетворити назад в числовий вектор, що складається з порядкових номерів рівнів факторів:

as.numeric (treatment)

[1] 2 2 2 2 2 2 1 1 1 1

Існує також спеціальна команда для створення факторів:

gl (n, k, length = n * k, labels = 1: n),

де n – кількість рівнів фактора; k – число повторів для кожного рівня; length – розмір підсумкового об'єкта; labels – необов'язковий аргумент, який можна використовувати для зазначення назв кожного рівня фактора. Наприклад, виконання наступної команди призведе до створення вектора my.fac, що є фактором з двома рівнями – Control і Treatment, причому кожна з міток "Control" і "Treatment" буде повторена по 8 разів:

my.fac = gl (2, 8, labels = c ("Control", "Treatment"))

my.fac

[1] Control Control Control Control Control Control Control

[8] Control Treatment Treatment Treatment Treatment Treatment Treatment

[15] Treatment Treatment

Levels: Control Treatment

Ще одна корисна команда створює фактори, розділивши область варіації числового вектора x на інтервали:

cut (x, breaks, labels),

де в якості аргументу breaks може виступати або необхідну кількість інтервалів, або вектор, що містить список "точок розриву", а labels визначає назви рівнів:

```
x <- c (1,2,3,4,5,2,3,4,5,6,7)
cut (x, breaks = 3)
[1] (0.994,3] (0.994,3] (3,5] (3,5] (3,5] (0.994,3] (3,5]
[8] (3,5] (3,5] (5,7.01] (5,7.01]
Levels: (0.994,3] (3,5] (5,7.01]
cut (x, breaks = 3, labels = letters [1: 3])
[1] a a b b b a b b b c c
Levels: a b c
cut (x, breaks = quantile (x, c (0, .25, .50, .75,1)),
labels = c ( "Q1", "Q2", "Q3", "Q4"), include.lowest = TRUE)
[1] Q1 Q1 Q2 Q2 Q3 Q1 Q2 Q2 Q3 Q4 Q4
Levels: Q1 Q2 Q3 Q4
```

У третьому фрагменті коду числової вектор "розрізаний" по квартильним значенням, а параметр include.lowest вказано, щоб уникнути появи невизначеності "NA" для значення x = 1.

На відміну від вектора або матриці, які можуть містити дані тільки одного типу, в список (list) або таблицю (data frame) можна включати поєднання будь-яких типів даних. Це дозволяє в одному об'єкті, зберігати різнорідну інформацію. Кожен компонент списку може бути змінною, вектором, матрицею, фактором або іншим списком. Крім того, ці елементи можуть належати до різних типів: числа, рядки символів, булеві змінні.

Списки є найбільш загальним засобом зберігання внутрісистемної інформації: зокрема, результати більшості статистичних аналізів в програмі R зберігаються в об'єктах-списках. Для створення списків в R служить однойменна функція list ().

Створимо три різнотипних вектори – з текстовими, числовими і логічними значеннями:

```
vector1 <- c ( "A", "B", "C")
vector2 <- seq (1, 3, 0.5)
vector3 <- c (FALSE, TRUE)
# об'єднаємо ці три вектори в один об'єкт-список, компонентам якого дамо імена Text, Number і Logic:
my.list <- list (Text = vector1, Number = vector2, Logic = vector3)
# переглянемо вміст створеного списку:
my.list
$ Text
[1] "A" "B" "C"
$ Number
[1] 1.0 1.5 2.0 2.5 3.0
$ Logic
[1] FALSE TRUE
```

До елементів списку можна отримати доступ за допомогою трьох різних операцій індексації. Для звернення до іменованих компонентів застосовують знак $. Так, для вилучення компонентів Text, Number і Logic зі створеного нами списку my.list необхідно послідовно ввести наступні команди:

```
my.list $ Text
[1] "A" "B" "C"
my.list $ Number
[1] 1.0 1.5 2.0 2.5 3.0
my.list $ Logic
[1] FALSE TRUE
```

Є можливість вставляти зі списку не тільки його зазначені компоненти-вектори, а й окремі елементи, що входять до ці вектори. Для цього необхідно скористатися вже розглянутим раніше способом – індексацією за допомогою квадратних дужок.

Особливість роботи зі списками полягає у тому, що спочатку необхідно вказати ім'я компоненти списку, використовуючи знак $, а вже потім номер (номери) окремих елементів цієї компоненти:

```
my.list $ Text [2]
[1] "B"
my.list $ Number [3: 5]
[1] 2.0 2.5 3.0
my.list $ Logic [1]
[1] FALSE
```

Витягування компонентів списку можна здійснювати також з використанням подвійних квадратних дужок для номеру компоненти списку:

```
my.list [[1]]
[1] "A" "B" "C"
my.list [[2]]
[1] 1.0 1.5 2.0 2.5 3.0
my.list [[3]]
[1] FALSE TRUE
```

Після подвійних квадратних дужок з індексним номером компоненти списку можна також вказати номер (номери) окремих елементів цієї компоненти:

```
my.list [[1]] [2]
[1] "B"
my.list [[2]] [3: 5]
[1] 2.0 2.5 3.0
my.list [[3]] [1]
[1] FALSE
```

Створений список my.list містив всього лише три невеликих вектора, і ми знали, які це вектори, і на якому місці в списку вони стоять. Однак на практиці можна зіткнутися зі списками, індексування яких може бути ускладнене через відсутність уявлень про їх структуру. Для з'ясування структури об'єктів в мові R є спеціальна функція str () (від structure):

```
str (my.list)
List of 3
$ Text: chr [1: 3] "A" "B" "C"
$ Number: num [1: 5] 1 1.5 2 2.5 3
$ Logic: logi [1: 2] FALSE TRUE
```

З наведеного прикладу випливає, що список my.list включає 3 компоненти (List of 3) з іменами Text, Number і Logic (перераховані в окремих рядках після знаку $). Ці компоненти відносяться до символьного (chr), числовому (num) і логічного (Logic) типам векторів відповідно. Крім того, команда str () виводить на екран перші кілька елементів кожного вектора.

Фрейм даних (data frame) являє собою об'єкт R, за структурою нагадує лист електронної таблиці Microsoft Excel. Кожен стовпець таблиці є вектором, що містить дані певного типу. При цьому діє правило, згідно з яким всі стовпці повинні мати однакову довжину (з "точки зору" R таблиця даних є окремим випадком списку, в якому все компоненти-вектори мають однаковий розмір).

Таблиці (фрейми) даних – це основний клас об'єктів R для зберігання даних.

Зазвичай такі таблиці готуються за допомогою зовнішніх програм (наприклад, Microsoft Excel) та потім завантажуються в середовище R. Проте невелику таблицю можна зібрати з декількох векторів засобами R. Для цього використовують функцію data.frame (). Припустимо, у нас є спостереження за загальною чисельністю чоловічого (Male) та жіночого (Female) населення в трьох містах City1, City2, і City3. Уявімо ці дані у вигляді однієї таблиці з ім'ям CITY. Для початку створимо текстові вектори з назвами міст (City) і статі (sex), а також вектор зі значеннями чисельності представників кожної статі (number):

```
city <- c ( "City1", "City1", "City2", "City2", "City3", "City3")
sex <- c ( "Male", "Female", "Male", "Female", "Male", "Female")
number <- c (12450, 10345, 5670, 5800, 25129, 26000)
```

Об'єднаємо ці три вектори в одну таблицю даних:

```
CITY <- data.frame (City = city, Sex = sex, Number = number)
CITY
```

```
     City    Sex     Number
1  City1  Male     12450
2  City1  Female  10345
3  City2  Male      5670
4  City2  Female   5800
5  City3 Male       25129
6  City3 Female    26000
```

В якості заголовків стовпців можуть виступати будь-які призначені для користувача імена, задовольняють вимогам R. Витягти окремі компоненти таблиць для виконання необхідних обчислень, як і в прикладах зі списками, можна з використанням знака $, квадратних дужок з зазначенням двох індексів [<номер_рядка>, номер_стовпчика>], подвійних квадратних дужок [[]], або безпосередньо на ім'я стовпця:

```
CITY $ Sex
[1] Male Female Male Female Male Female
Levels: Female Male
CITY $ Number
[1] 12450 10345 5670 5800 25129 26000
# Ідентичні результати можна отримати за допомогою команд:
CITY [, 2]
[1] Male Female Male Female Male Female
Levels: Female Male
CITY [[3]]
[1] 12450 10345 5670 5800 25129 26000
CITY [ "Sex"]
[1] Male Female Male Female Male Female
Levels: Female Male
CITY [ "Number"]
[1] 12450 10345 5670 5800 25129 26000
```

Після імені або індексного номера стовпця можна вказувати індексні номери окремих комірок таблиці, що дозволяє витягувати вміст цих комірок:

```
# Витягуємо 4-й елемент з стовпця Number:
CITY $ Number [4]
[1] 5800
# Витягуємо елементи 1-3 з стовпця Number:
CITY $ Number [1: 3]
[1] 12450 10345 5670
# Витягуємо всі значення чисельності, що перевищують 10000
CITY $ Number [CITY $ Number> 10000]
[1] 12450 10345 25129 26000
# Витягуємо всі значення чисельності чоловічого населення:
CITY $ Number [CITY $ Sex == "Male"]
[1] 12450 5670 25129
# Повторюємо ті ж команди, але з використанням []:
CITY [4, 3]
[1] 5800
CITY [1: 3, 3]
[1] 12450 10345 5670
CITY [CITY $ Number> 10000, 3]
[1] 12450 10345 25129 26000
CITY [CITY $ Sex == "Male", 3]
[1] 12450 5670 25129
```

При роботі з великими таблицями даних буває складно візуально досліджувати їх вміст перед початком аналізу. Повну зведену інформацію про таблиці (так само як і про інші об'єкти R) можна отримати за допомогою функції str ():

```
str (CITY)
'Data.frame': 6 obs. of 3 variables:
$ City: Factor w / 3 levels "City1", "City2", .. 1 1 2 2 3 3
```

$ Sex: Factor w / 2 levels "Female", "Male": 2 1 2 1 2 1

$ Number: num 12450 10345 5670 5800 25129 ...

У представленому звіті об'єкт CITY є таблицею даних, в склад якої входять три змінні з шістьма спостереженнями кожна. Дві з цих змінних – City і Sex програма автоматично розпізнала як чинники з трьома і двома рівнями відповідно. Змінна Number є кількісною. Для зручності виводяться також кілька перших значень кожної змінної. Часто виникає необхідність з'ясувати лише імена змінних, що входять в таблицю даних. Це можна зробити за допомогою команди names ():

names (CITY)

[1] "City" "Sex" "Number"

Є також можливість швидко переглянути кілька перших або кілька останніх значень кожної змінної, що входить до складу таблиці даних. Для цього використовуються функції head () і tail () відповідно:

head (CITY, n = 3)

City Sex Number

1 City1 Male 12450

2 City1 Female 10345

3 City2 Male 5670

tail (CITY, n = 2)

City Sex Number

5 City3 Male 25129

6 City3 Female 26000

При необхідності внесення виправлень в таблицю можна скористатися вбудованим в R редактором даних. Зовні цей редактор нагадує звичайний лист Excel, однак має дуже обмежені функціональні можливості. Все, що він дозволяє робити – це додавати нові або виправляти вже введені значення змінних, змінювати заголовки стовпців, а також додавати нові рядки і стовпці.

Працюючи в стандартній версії R, редактор даних можна запустити з меню "Файли>Редактор даних", або виконавши команду fix () (fix – виправляти) з командного рядка консолі R (наприклад, fix (CITY)). Після внесення виправлень редактор просто закривають – усі зміни будуть збережені автоматично.

Заповнення порожніх значень

Часто на практиці деякі значення в таблиці відсутні, що може бути зумовлено великою кількістю причин: на момент вимірювання прилад вийшов з ладу, по неуважності персоналу вимір не було занесено в протокол дослідження, випробуваний відмовився відповідати на певне питання в анкеті, була загублена проба тощо. Комірки з такими відсутніми значеннями (missing values) в таблицях даних R не можуть бути просто порожніми – інакше стовпці таблиці виявляться різної довжини. Для позначення відсутніх спостережень в мові R, як вказувалося раніше, є спеціальне значення – NA (not available – не доступно). Якщо значення NA має сенс нуля (наприклад, примірників деякого виду виявлено не було), то легко зробити цю заміну в таблиці командою DF:

```
DF [is.na (DF)] <- 0
```

Сортування таблиць

Для сортування рядків таблиці по різними ключами використовується функція order ():

```
DF <- data.frame (X1 = c (1,15,1,3), X2 = c (1,0,7,0), X3 = c (1,0,1,2),
X4 = c (7,4,41,0), X5 = c (1,0,5,3))
row.names (DF) <- c ( "A", "B", "C", "D")
# DF1 – таблиця, стовпці якої відсортовані за спаданням суми значень
DF1 <- DF [, rev (order (colSums (DF)))]
# DF2 – таблиця, рядки якої відсортовані в висхідному порядку по 1 стовпчику, потім в низхідному по другому
DF2 <- DF [order (DF $ X1, -DF $ X2),]
```

Об'єднання таблиць

Нехай ми маємо дві таблиці:

DF1				
Y	N	A	B	C
12	22	0	1	0
12	23	1	3	0
12	24	0	0	1

DF2				
Y	N	A	B	D
13	22	0	1	2
13	23	0	3	0
13	24	1	0	5

Об'єднати їх стовпці можна з використанням функції cbind ():

cbind (DF1, DF2)

Y N A B C Y N A B D

1 12 22 0 1 0 13 22 0 1 2

2 12 23 1 3 0 13 23 0 3 0

3 12 24 0 0 1 13 24 1 0 5

Для об'єднання рядків ми повинні попередньо перетворити таблиці до єдиного списку стовпців:

DF1 [, names (DF2) [! (Names (DF2)% in% names (DF1))]] <- NA

DF2 [, names (DF1) [! (Names (DF1)% in% names (DF2))]] <- NA

rbind (DF1, DF2)

Y N A B C D

1 12 22 0 1 0 NA

2 12 23 1 3 0 NA

3 12 24 0 0 1 NA

4 13 22 0 1 NA 2

5 13 23 0 3 NA 0

6 13 24 1 0 NA 5

Аналогічну операцію ми можемо виконати за допомогою команди merge(DF1, DF2, all = TRUE). Функція merge () дозволяє виконувати об'єднання таблиць усіма поширеними способами join-операцій мови SQL.

Висновок до лекції 11

R – це популярна мова програмування, що використовується для статистичних обчислень та графічного представлення даних. Найбільш поширеним використанням є аналіз та візуалізація даних, машинне навчання.

R забезпечує багато статистичних методів (статистичні тести, класифікація, кластеризація даних тощо).

Мова R працює на різних платформах (Windows, Mac, Linux), є відкритою і безкоштовною, має велику підтримку громади, містить багато пакетів (бібліотек функцій), які можна використовувати для вирішення різних задач з аналізу даних.

Питання для закріплення

1. Яка історія розвитку мови R?
2. Які можливості мови R?
3. Що таке об'єкти, пакети, функції в R?
4. Як утворюються вектори та матриці в R? Як виконувати операції над ними в в R?
5. Що таке фактори? Як визначаються списки, фрейми та дії над ними в R?

Список рекомендованої літератури

1. Microsoft R Application Network // Електронний ресурс. Режим доступу: https://mran.microsoft.com/documents/what-is-r

2. R програмування // Електронний ресурс. Режим доступу: https://coderlessons.com/tutorials/mashinnoe-obuchenie/r-programmirovanie/r-programmirovanie

3. R Introduction // Електронний ресурс. Режим доступу: https://www.w3schools.com/r/r_intro.asp

Лекція 12.
Експорт, імпорт та оброблення даних в R

План лекції

12.1. Експорт та імпорт даних в R.
12.2. Використання R для аналізу часових рядів.
12.3. Оброблення даних в R.

12.1. Експорт та імпорт даних в R

Розглянемо найбільш поширені способи імпорту таблиць даних в робоче середовище R, проте спочатку варто ознайомитися з правилами підготовки завантаження:

– в імпортованої таблиці з даними не повинно бути порожніх клітинок, якщо деякі значення з тих чи інших причин відсутні, замість них слід ввести NA.

– імпортовану таблицю з даними рекомендується перетворити в простий текстовий файл з одним з допустимих розширень. На практиці зазвичай використовуються файли з розширенням .txt, в яких значення змінних розділені знаками табуляції (tab-delimited files), а також файли з розширенням .csv (comma separated values), в яких значення змінних розділені комами або іншим символом.

– у якості першого рядка імпортованої таблиці рекомендується ввести заголовки стовпців-змінних. Якщо такий рядок відсутній, то про це необхідно повідомити в описі команди, яка буде керувати завантаженням файлу (наприклад, **read.table ()**). Усі наступні рядки файлу в якості першого елемента можуть містити заголовки рядків (якщо такі передбачені), після яких слідують значення кожної з наявних в таблиці змінних.

Імена стовпців таблиці краще привласнити з дотриманням правил ідентифікації змінних R, тобто виключити прогалини та інші спеціальні символи, крім крапки і підкреслення. Щоб уникнути проблем, пов'язаних з кодуванням, всі текстові величини в імпортованих файлах варто створювати з використанням літер латинського алфавіту.

Файл для імпортування рекомендується помістити в робочу папку програми, тобто папку, в якій R за замовчуванням буде "намагатися знайти" цей файл. Щоб з'ясувати шлях до робочої папки R на своєму комп'ютері використовується команда **getwd ()** (get working directory – дізнатися робочу директорію):

```
getwd ()
[1] "C: / Temp /"
```

Змінити робочу директорію можна за допомогою команди **setwd ()** (set working directory – встановити робочу директорію):

setwd ("C: / My Documents")

При виконанні цієї команди зовні нічого не станеться, однак подальше застосування команди **getwd ()** покаже, що шлях до робочої папки змінився:

getwd ()

[1] "C: / My Documents /"

Нижче наведено фрагмент типової таблиці даних, яка може бути успішно завантажена для аналізу в середовище R:

	Group	Variable1	Variable2	Variable3
Ivan	A	102	1.3	14
Vitaliy	A	98	1.4	11
Oleg	B	45	NA	8
Mikhail	B	50	3.2	6

Як бачимо, наведений фрагмент має розмірність 5×5, тобто складається з п'яти рядків і п'яти стовпців. У першому рядку представлені заголовки всіх наявних в таблиці стовпців, за винятком першого. Перший стовпець, хоча і не має власного заголовка, не є порожнім – він містить імена добровольців, які брали участь у деякому експерименті (Ivan, Vitaliy і т.д.). Другий стовпець має заголовок Group і містить мітки, за якими можна з'ясувати приналежність випробовуваних до тієї чи іншої експериментальної групи (A, B і т.д.). У термінах мови R змінна Group називається фактором. У наступних стовпчиках (з заголовками Variable1, Variable2 і т.д.) містяться значення виміряних в ході дослідження змінних. У наведеному фрагменті таблиці є одне відсутнє значення, замість якого введено NA.

Одним з найбільш доступних засобів підготовки даних для їх подальшого аналізу за допомогою R є програма Microsoft Excel. Для збереження Excel-таблиць у вигляді txt- або csv-файлів зазвичай пропонують використовувати опцію Зберегти як (Save as) в розділі Файл (File) головного меню цієї програми. Іншим простим і надійним способом експорту даних з Excel є створення в редакторі Блокнот (Notepade) нового файлу і перенесення туди через буфер обміну всієї таблиці або виділеної її частини.

Основною функцією для імпортування даних в робоче середовище R є **read.table ()**. Ця потужна функція дозволяє налаштувати процес завантаження зовнішніх файлів, в зв'язку з чим, вона має велику кількість керуючих аргументів (табл. 12.1.).

Таблиця. 12.1. Керуючі аргументи функції read.table ()

Аргумент	Призначення
file	Служить для вказівки шляху до імпортованого файлу. Шлях приводять або в абсолютному вигляді (наприклад, file = "C: /Temp/MyData.dat"), або вказують тільки ім'я файлу, що імпортується (наприклад, file ="MyData.txt"), але за умови, що останній зберігається в робочій папці програми. Як ім'я можна вказувати URL-посилання на файл, який передбачається завантажити з мережі (наприклад: file = "http://somesite.net/YourData.csv"). Починаючи з версії R 2.10, з'явилася можливість імпортувати архівовані файли в zip-форматі.
header	Служить для повідомлення програмі про наявність в завантаженні рядків з заголовками стовпців. За замовчуванням приймає значення FALSE. Якщо рядок з заголовками стовпців є, цього аргументу слід привласнити значення TRUE.
row.names	Служить для вказівки номера стовпчика, у якому містяться імена рядків (наприклад, в розглянутому вище прикладі це був перший стовпець, тому row.names = 1). Важливо пам'ятати, що всі імена рядків повинні бути унікальними, тобто однакові імена для двох або більше рядків не допускаються.

sep	дозволяє вказати роздільник значень змінних, що використовується у файлі (separator – роздільник). За замовчуванням передбачається, що значення змінних розділені "порожнім простором", наприклад, у вигляді пробілу або знаку табуляції (sep = ""). У файлах формату csv значення змінних розділені комами, і тому для них sep = ",".
dec	Служить для вказівки знака, що використовується у файлі для відділення цілої частини числа від дробу. За замовчуванням dec = ".". У багатьох країнах застосовують кому, про що важливо згадати перед завантаженням файлу і, при необхідності, використовувати dec = ",". Потрібно слідкувати, щоб dec і sep не були б однаковими.
nrows	Виражається цілим числом, що вказує кількість рядків, яке має бути зчитане з завантажуваної таблиці. Від'ємні та інші значення ігноруються. Приклад: nrows = 100.
skip	Виражається цілим числом, що вказує кількість рядків у файлі, яке повинно бути пропущено перед початком імпортування. Приклад: skip = 5

Для завантаження підготовлених файлів досить використовувати мінімальний набір аргументів функції **read.table ()**. Потрібно завантажити файл hydro_chem.txt, який зберігається у робочій папці R та містить дані про хімічний склад води водоймища. Завантажуємо таблицю даних, які ми хочемо зберегти у вікні об'єкта з ім'ям chem:

chem <- read.table (file= "hydro_chem.txt", header= TRUE)

Імпортовані в R файли зазвичай мають формат csv. Для їх завантаження можна використати функцію read.table (), слід вказати символ, який

використовується у якості роздільника значень змінних у файлі (наприклад, кома):

chem <- read.table (file= "hydro_chem.csv", header= TRUE, sep = ",")

Аналогом read.table () для зчитування csv-файлів є функція read.csv ():

chem <- read.csv (file= "hydro_chem.csv", header= TRUE)

Якщо файл для завантаження зберігається у папці, відмінній від робочої папки R, то слід вказати повний шлях до нього. Користувачам операційної системи Windows необхідно пам'ятати, що для вказівки повних шляхів до файлів в програмі R використовується не зворотний одинарний слеш (\), а прямий одинарний (/) або подвійний зворотний слеш (\\). Наприклад, наступні дві команди будуть успішно сприйняті R і приведуть до ідентичного результату – завантаження файлу hydro_chem.txt і збереження його у вигляді об'єкта chem:

chem <- read.csv (file = "D: \\ Documents \\ hydrochem.txt", header = TRUE)

chem <- read.csv (file = "D: /Documents/hydrochem.txt", header = TRUE)

Для інтерактивного вибору завантаження, який зберігається поза робочою папкою R, можна застосувати допоміжну функцію file.choose () (вибрати файл). Виконання цієї команди призводить до відкриття діалогового вікна операційної системи Windows, в якому користувач вибирає папку з необхідним файлом. Зручно поєднувати file.choose () з командами read.table () або read.csv (), наприклад [1]:

chem <- read.table (file = file.choose (), header = TRUE, sep = ",").

12.2. Використання R для аналізу часових рядів

У R існує спеціальний клас об'єктів для роботи з часовими рядами – **ts** (від time series – часовий ряд). Для створення об'єктів цього класу служить однойменна функція **ts ()**. Розглянемо щомісячні дані з народжуваності в м Нью-Йорк, зібрані в період з січня 1946 по грудень 1959 р [2]:

birth <-scan ("http://robjhyndman.com/tsdldata/data/nybirths.dat")

Об'єкт birth є вектором з 168 щомісячними значеннями народжуваності (в тис. чоловік), в чому можна переконатися за допомогою функції:

is.vector (birth)

[1] TRUE

Функція head () дозволяє переглянути перші кілька значень вектора birth (за замовчуванням перші 6 значень):

head (birth)

[1] 26.663 23.598 26.931 24.740 25.806 24.364

Перетворити об'єкт birth в часовий ряд дуже просто:

birth.ts <- ts (birth, start = c (1946 1), frequency = 12)

Аргумент start був використаний для того, щоб вказати дату, з якої починається часовий ряд birth.ts (1946 рік, 1-й місяць). Додатковий аргумент frequency (частота) дозволяє задати крок збільшення наступних дат – у розглянутому прикладі рік розбивається на 12 проміжків, крок приросту складає 1 місяць. Створений таким чином об'єкт birth.ts при перегляді зовні нагадує матрицю. При цьому рядкам і стовпцям цієї матриці були автоматично присвоєні відповідні імена, виходячи з значень аргументів start і frequency, (дані за стовпцями Oct, Nov, Dec опущені):

birth.ts

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
1946	26.663	23.598	26.931	24.740	25.806	24.364	24.477	23.901	23.175
1947	21.439	21.089	23.709	21.669	21.752	20.761	23.479	23.824	23.105
1948	21.937	20.035	23.590	21.672	22.222	22.123	23.950	23.504	22.238
1949	21.548	20.000	22.424	20.615	21.761	22.874	24.104	23.748	23.262
1950	22.604	20.894	24.677	23.673	25.320	23.583	24.671	24.454	24.122
1951	23.287	23.049	25.076	24.037	24.430	24.667	26.451	25.618	25.014
1952	23.798	22.270	24.775	22.646	23.988	24.737	26.276	25.816	25.210
1953	24.364	22.644	25.565	24.062	25.431	24.635	27.009	26.606	26.268
1954	24.657	23.304	26.982	26.199	27.210	26.122	26.706	26.878	26.152
1955	24.990	24.239	26.721	23.475	24.767	26.219	28.361	28.599	27.914
1956	26.217	24.218	27.914	26.975	28.527	27.139	28.982	28.169	28.056
1957	26.589	24.848	27.543	26.896	28.878	27.390	28.065	28.141	29.048
1958	27.132	24.924	28.963	26.589	27.931	28.009	29.229	28.759	28.405
1959	26.076	25.286	27.660	25.951	26.398	25.565	28.865	30.000	29.261

Функція **is.ts ()** дозволяє перевірити, чи дійсно створений нами об'єкт birth.ts є часовим рядом:

is.ts (birth.ts)

[1] TRUE

В R є досить великий набір методів для роботи з об'єктами класу ts. Зокрема, за допомогою функції plot () можна швидко зобразити часовий ряд графічно:

plot (birth.ts, xlab = "", ylab = "")

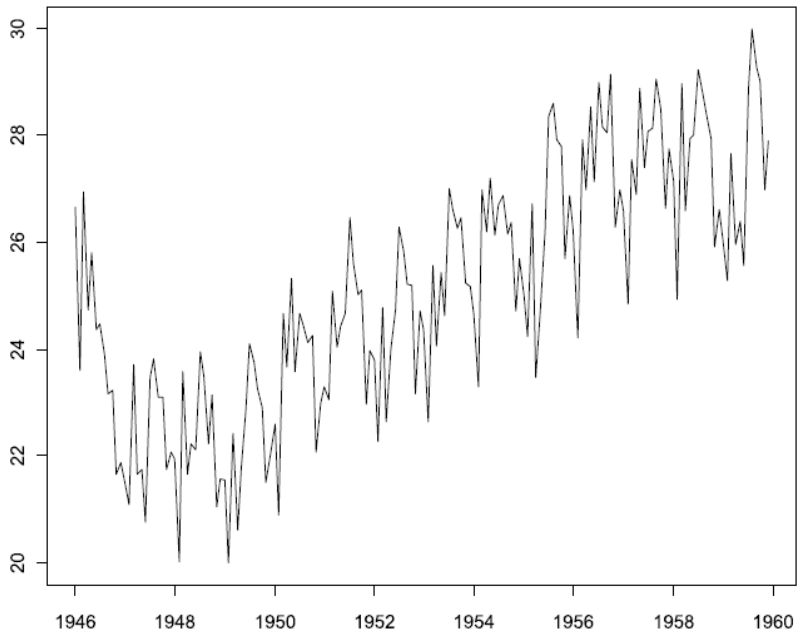


Рис. 12.1. Графічне представлення часового ряду [2]

12.3. Оброблення даних в R

Більшість процедур оброблення даних в R реалізується за допомогою функцій. Функції – це іменований програмний код, що складається з деякого набору змінних, констант, операторів та інших функцій, призначений для виконання конкретних операцій і завдань. Як правило (але не завжди), функції повертають результат свого виконання у вигляді об'єкта мови R – змінної певного класу: вектора, списку, таблиці тощо. За своїм призначенням функції можна розділити на характерні групи: арифметичні, символьні, статистичні та інші. Функції можуть бути вбудованими (представленими в базових або підвантажуваних пакетах) і власними (написаними безпосередньо самими користувачами).

Трьома характерними рисами мови R як мови високого рівня є модульність побудови, орієнтація на об'єкти та векторизація обчислень. Під модульністю розуміється широке використання груп виразів і функцій. Вирази expr, що складаються з об'єктів даних, викликів функцій та інших операторів мови можуть групуватися в фігурних дужках: {expr_1; ...; expr_m}, і значення, яке повертає ця група, являє собою результат виконання останнього виразу. Оскільки така група є також виразом, то вона може бути, наприклад, включена в круглі дужки і використовуватися як частина більш загального виразу.

Група команд нижче виконує розрахунок середнього і стандартного відхилення натурального ряду чисел від 1 до 10 і повертає вектор з цих значень:

{Aver <- mean (1:10); stdev <- sd (1:10); c (MEAN = aver, SD = stdev)}

MEAN SD

5.50000 3.02765

Якщо цей розрахунок потрібно виконати багато разів для різних наборів вихідних даних, то його потрібно оформити у вигляді функції. Загальний синтаксис оформлення власної функції користувача такий:

ім’я_функції <- function (argl, arg2, ...) {

група_виразів

return (object)}

де ім’я_функції – ім'я створюваної функції, argl, arg2, ... – формальні аргументи функції. Оператор return () потрібен у випадках, коли група виразів не повертає цільового результату.

Перед своїм першим виконанням функція повинна бути визначена в поточному скрипті або завантажена за допомогою команди source () з скриптового файлу, де вона була попередньо підготовлена.

Тоді виклик функції може бути здійснений як ім'я_функції (argl, arg2, ...), де argl, arg2, ... – фактичні аргументи, пов'язані з формальними параметрами функції по порядку їх слідування, або за найменуваннями.

Для представленого вище прикладу можна оформити функцію:

stat_param <- function (x) {

aver <- mean (x); stdev <- sd (x); c (MEAN = aver, SD = stdev)}

і включити її в колекцію власних функцій, розташованих в файлі my_func.R.

Тоді необхідний нам результат, наведений вище, можна отримати, виконавши

source ("my_func.R")

stat_param (1:10)

Компоненти списку аргументів в заголовку функцій можуть бути обов'язковими або приймати опціональні значення. Наприклад, наступна функція підносить числовий об'єкт x в степінь n, але якщо степінь не вказана, то автоматично відбувається зведення в куб:

power <- function (x, n = 3) {x ^ n}

Аргументами функцій можуть бути об'єкти різного типу, наприклад, назви інших функцій. Так, наступна функція виконує довільні перетворення випадкових рівномірно розподілених величин:

my_exampl <- function (n, func_trans)

{X <- runif (n); abs (func_trans (x))}

У мові R широко використовуються розгалуження і цикли обчислювального процесу. Умовний оператор має наступну структуру:

if (логічний_вираз)

{група_виразів_1 якщо логічний_вираз TRUE}

else {група_виразів_2 в іншому випадку}

Наприклад, наступна функція порівнює розміри двох векторів:

compare <- function (x, y) {nl <- length (x); n2 <- length (y)

if (nl! = n2) {

if (nl> n2) {z = (nl - n2)

cat ("Перший вектор має на", z, "елементів більше \ n")}

else {z = (n2 - nl)

cat ("Другий вектор має на", z, "елементів більше \ n")}}

else {cat ("Кількість елементів однакова", nl, "\ n")}

}

x <- з (1: 4)

y <- з (1: 9)

compare (x, y)

Є також скорочена форма реалізації розгалужень:

ifelse (логічний_вираз, група_виразів_1, група_виразів_2).

Повторення в циклі одних і тих же обчислювальних операцій здійснюється з використанням конструкцій for (), while () або repeat (), які мають наступний синтаксис:

for (index in for_object) {група_виразів}

while (логічний_вираз) {група_виразів}

repeat {група_виразів; break}

Об'єкт for_object може бути вектором, масивом, таблицею, або списком, а група_виразів виконується кожен раз для кожного елемента index цього об'єкта.

У мові R замість того, щоб виконувати послідовно скалярні операції над кожним з елементів масиву, набагато ефективніше виконувати паралельні обчислення, при яких програма обробляє одночасно увесь масив (вектор) цілком або по кілька елементів вектора в кожен момент часу. Очевидно, що такий підхід потенційно може привести до значного прискорення однотипних обчислень над великими масивами даних.

Розглянемо найпростіший приклад векторизованих обчислень в R. Припустимо, у нас є вектор з 10 додатних чисел і ми хочемо знайти квадратний корінь з кожного з них. Замість написання циклу для почергового виконання цієї операції над кожним елементом, цей вектор подається на вхід функції sqrt (), яка повертає вектор з результатами обчислень:

x <- 1:10

sqrt (x)

[1] 1.000 1.414 1.732 2.000 2.236 2.449 2.645 2.828 3.000 3.162

Принцип векторизованих обчислень можна застосувати не тільки до векторів, а й до більш складних об'єктів R – матриць, списків і таблиць даних

(для R різниці між останніми двома типами об'єктів не існує: фактично таблиця даних є списком з декількох компонентів – векторів однакового розміру).

У базовій комплектації R є сімейство функцій, призначених для організації векторизованих обчислень над такими об'єктами. У назві цих функцій є слово apply (*англ*. застосувати), якому передує літера, яка вказує на принцип роботи тієї чи іншої функції, при цьому.

Apply () на відміну від for () можна легко розпалалелити (перейменувавши функцію в її паралельну версію з пакета snow);

– алгоритми, записані без циклів, легше модифікуються, містять менше помилок і легше набираються в командному рядку;

– результат роботи apply () може бути аргументом функції, будь-яка функція може бути вставлена в якості аргументу в apply ().

Функція **apply ()** використовується у випадках, коли необхідно застосувати деяку функцію до всіх рядків або стовпців матриці (або масивів більшої розмірності):

apply (x, MARGIN, FUN, ...)

де x – це перетворений об'єкт, MARGIN – індекс, що позначає напрямок процесу обчислень (по стовпцях або рядках), FUN – використовувана для обчислень функція, ... – це будь-які інші параметри застосовуваної функції. Для матриці або таблиці даних MARGIN = 1 позначає рядки, а MARGIN = 2 – стовпці. Оскільки FUN означає будь-яку функцію R, то функція apply () – це потужний засіб модульного оброблення даних.

Створимо двовимірну матрицю:

M <- matrix (seq (1,16), 4, 4)

Знайдемо мінімальні значення в кожному рядку матриці:

apply (M, 1, min)

[1] 1 2 3 4

Знайдемо мінімальні значення в кожному стовпці матриці:

apply (M, 2, max)

[1] 4 8 12 16

Приклад з тривимірним масивом:

M <- array (seq (32), dim = c (4,4,2))

Застосуємо функцію sum () до кожного елементу M [*,,], тобто виконаємо підсумовування по вимірами 2 і 3:

apply (M, 1, sum)

Результат – одновимірний вектор:

[1] 120 128 136 144

Застосуємо функцію sum () до кожного елементу M [*, *,], тобто виконаємо підсумовування по третього виміру:

apply (M, c (1,2), sum)

Результат – матриця:

[, 1] [2] [3] [4]

[1,] 18 26 34 42

[2,] 20 28 36 44

[3,] 22 30 38 46

[4,] 24 32 40 48

При необхідності обчислення сум і середніх значень за рядками або стовпчиками матриць рекомендується використовувати швидкі і спеціально оптимізовані для цього функції colSums (), rowSums (), colMeans і rowMeans ().

Функція **lapply ()** використовується у випадках, коли необхідно застосувати функцію до кожного компоненту списку і отримати результат у вигляді списку (буква "l" в назві lapply () означає list – "список").

Функція **sapply ()** використовується у випадках, коли необхідно застосувати певну функцію до кожного компоненту списку, але результат вивести у вигляді вектора (буква "s" в назві sapply () означає simplify – "спростити").

Список з трьох компонентів:

x <- list (a = 1, b = 1: 3, c = 10: 100)

З'ясуємо розмір кожного компонента списку x:

sapply (x, FUN = length)

a b c

1 3 91 # результат повернений у вигляді вектора

Підсумовування усіх елементів в кожному компоненті списку x:

sapply (x, FUN = sum)

a b c

1 6 5005 # результат повернений у вигляді вектора.

Функція **replicate ()** є свого роду "обгорткою" для функції sapply () і дозволяє провести обчислення з метою генерації набору чисел згідно заданого алгоритму. Синтаксис функції має вигляд:

replicate (n, expr, simplify = TRUE)

де n – число повторів, expr – функція або група виразів, які треба повторити n раз, simplify = TRUE – необов'язковий параметр, який спрощує результат і представляє його у вигляді вектора або матриці значень.

Функція **vapply ()** схожа на sapply (), але працює швидше за рахунок того, що користувач однозначно вказує тип значень, що повертаються (буква "v" в назві vapply () означає velocity – "швидкість"). Такий підхід дозволяє уникати повідомлень про помилки (і переривання обчислень), що виникають при роботі з sapply () у деяких ситуаціях.

Функція **mapply ()** використовується у випадках, коли необхідно поелементно застосувати будь-яку функцію одночасно до декількох об'єктів. Результат повертається у вигляді вектора або масиву іншої розмірності. Буква "m" в назві mapply () означає multivariate – "багатовимірний" (одночасне виконання обчислень над елементами кількох об'єктів).

mapply (sum, 1: 5, 1: 5, 1: 5)

[1] 3 6 9 12 15

Функція **rapply ()** використовується у випадках, коли необхідно застосувати функцію до компонентів вкладеного списку (буква "r" в назві rapply () означає recursively – "рекурсивно").

Функція **tapply ()** використовується у випадках, коли необхідно застосувати будь-яку функцію fun до окремих груп елементів вектора x, заданим відповідно до рівнів будь-якого фактора group:

tapply (x, group, fun, ...)

У наступному прикладі функція **sample ()** використовується для створення двох випадкових вибірок: з 50 значень цілих чисел від 1 до 4 і пов'язаних з ними міток чотирьох груп A-D.

Функція tapply () підраховує суми x для кожного із значень фактора:

```
x <- sample (1: 4, size = 50, replace = T)
gr <- as.factor (sample (c ( "A", "B", "C", "D"), size = 50, replace = T))
tapply (x, gr, sum)
A B C D
29 25 30 37
```

Функція **by ()** є аналогом функції tapply (), що застосовується для таблиць. Таблиця data розділяється відповідно до заданого стовпця-фактора group на підмножини підтаблиць, для обробки кожної частини визначається функція fun:

by (data, group, fun, ...)

Функція дозволяє виконати комбінаторну операцію fun над елементами двох масивів або векторів x і y, не вдаючись до явного використання "подвійного" циклу:

outer (x, y, fun = "*", ...)

За замовчуванням здійснюється операція попарного перемноження:

```
x <- 1: 5; y <- 1: 5
outer (x, y)
   [, 1] [2] [3] [4] [5]
[1,]  1    2    3   4    5
[2,]  2    4    6   8    10
[3,]  3    6    9   12  15
[4,]  4    8    12  16  20
[5,]  5   10   15  20 25
```

Використовуючи функцію outer () разом з функцією paste (), можна згенерувати всі можливі попарні комбінації "зв'язок" елементів символьного і цілочисленого векторів:

```
x <- з ( "A", "B", "C", "D")
y <- 1:10
outer (x, y, paste, sep = "")
      [, 1]   [2]   [3]   [4]   [5]    [6]    [7]    [8]    [9]   [, 10]
[1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9" "A10"
[2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9" "B10"
[3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9" "C10"
[4,] "Dl" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9" "D10".
```

Висновок до лекції 12

Основною функцією для імпортування даних в робоче середовище R є функція read.table (), яка дозволяє налаштовувати процес завантаження зовнішніх файлів за допомогою керуючих аргументів. В R існує спеціальний клас об'єктів ts для роботи з часовими рядами, для створення об'єктів цього класу використовується функція ts ().

Більшість процедур оброблення даних в R реалізується за допомогою функцій. Характерними рисами мови R є модульність побудови, орієнтація на об'єкти та векторизація обчислень.

Мова R дозволяє розпаралелювати операції, щоб прискорити обчислення. Пакет для паралельних обчислень дозволяє виконувати завдання в різних ядрах машини.

Питання для закріплення

1. Як виконати експорт та імпорт даних в R?
2. Які інструменти мови R використовуються для аналізу часових рядів?
3. Як відбувається оброблення даних в R?
4. Для чого використовуються функції apply (), lapply (), sapply (), replicate (), mapply (), rapply (), tapply (), sample (), by (), outer ()?

Список рекомендованої літератури

1. Імпорт даних в R // Електронний ресурс. Режим доступу: https://r-analytics.blogspot.com/2011/07/r_24.html
2. Using R for Time Series Analysis // Електронний ресурс. Режим доступу: https://a-little-book-of-r-for-time-series.readthedocs.io/en/latest/src/timeseries.html
3. Why is `vapply` safer than `sapply`// Електронний ресурс. Режим доступу: https://stackoverflow.com/questions/12339650/why-is-vapply-safer-than-sapply

Лекція 13.
Основні інструменти аналізу та візуалізації даних в R

План лекції

13.1. Функція plot () та її параметри

Графіки є невід'ємною частиною розвідувального аналізу даних, оскільки дозволяють виявляти закономірності і тренди в складних наборах даних, а також можуть бути результатом статистичного аналізу (наприклад, при побудові дерев класифікації). Як правило, створення графіка починається з функції високого рівня, яка визначає його загальну структуру: розмірність (1D, 2D, 3D), масштаби осей, назви тощо.

Найбільш часто використовувані графічні функції високого рівня – plot (), hist (), boxplot (), scatterplot () і pairs (). З використанням багатого набору функцій низького рівня до побудованого графіку можуть бути додані додаткові елементи: текст, лінії, легенда тощо. (Прикладами таких функцій є lines (), points (), text () і axis ()).

Особливостями деталей графічного зображення управляє набір параметрів, як правило, загальний для більшості високорівневих і низькорівневих функцій, який визначає колір, типів і яких розмірів символів або маркерів, товщину і характер ліній, штрихування, рамку графіка тощо.

Функція **plot ()** – головна "робоча конячка", яка використовується для побудови графіків в R. Поведінка цієї функції високого рівня визначається

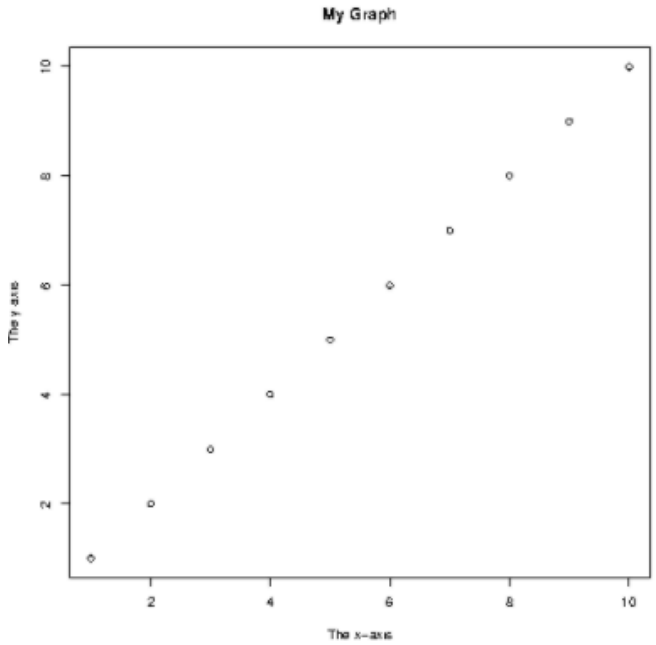
класом об'єктів, які вказуються в якості її аргументів. Відповідно, за допомогою plot () можна створити великий набір різнотипних графіків.

Функція plot () має велику кількість управляючих параметрів, які дозволяють здійснити тонке налаштування зовнішнього вигляду графіка.

1. Параметри xlab і ylab

Параметри **xlab** і **ylab** служать для зміни назв осей X і Y відповідно.

```
# We need this line of code to show graphs in our compiler
bitmap(file="out.png")

plot(1:10, main="My Graph", xlab="The x-axis", ylab="The y axis")
```



2. Параметр type

Параметр **type** дозволяє змінювати зовнішній вигляд точок на графіку. Він приймає одне з наступних значень:

- "p" – точки (points; використовується за замовчуванням);
- "l" – лінії (lines);
- "b" – зображуються і точки, і лінії (both points and lines);
- "o" – точки зображуються поверх ліній (points over lines);
- "h" – гістограма (histogram);
- "s" – ступінчаста крива (steps);
- "n" – дані не відображаються (no points).

3. Параметри xlim і ylim

Ці два параметри контролюють розмах значень на кожній з осей графіка. За замовчуванням вони обидва приймають значення NULL – у цьому випадку розмах вибирається програмою автоматично. Для скасування автоматичних налаштувань відповідному параметру необхідно присвоїти значення у вигляді числового вектора, що містить мінімальне і максимальне значення, які повинні відображатися на осі. Наприклад:

```
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", xlim = c (0, 15))
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", ylim = c (0, 5))
```

4. Параметри axes і ann

Ці два параметри контролюють відображення осей і їх назв відповідно. Кожен з них може приймати одне з двох можливих значень – TRUE або FALSE:

```
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", axes = TRUE, ann = TRUE)
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", axes = FALSE, ann = TRUE)
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", axes = TRUE, ann = FALSE)
```

5. Параметр log

За допомогою аргументу log можна перевести одну або обидві осі графіка на логарифмічну шкалу, наприклад:

```
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", log = "x")
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", log = "y")
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", log = "xy")
```

6. Параметр main

Аргумент main служить для створення заголовка графіка. За замовчуванням назва розміщується у верхній частині малюнка:

```
plot (indo.times, means, xlab = "Час", ylab = "Концентрація", main = "Швидкість виведення індометацину", type = "o")
```

13.2. Управління загальними параметрами - аргументами графічних функцій

Розглянемо графічні параметри, які контролюють зовнішній вигляд графіків, наприклад, тип, розмір і колір символів і ліній, тип і розмір шрифту у назвах графіка і його осей, використання математичних символів в назвах, розміщення легенди тощо. Вони застосовуються в якості аргументів не тільки при виклику plot (), але і багатьох інших функцій.

1. Тип символу

Змінити тип символів, які використовуються для відображення спостережень, дозволяє аргумент **pch** (plotting character – символ зображення). У стандартних випадках цей аргумент приймає чисельні значення від 1 до 25 (рис. 13.1).

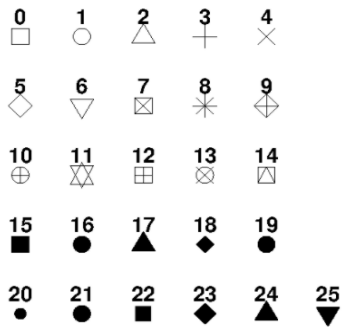
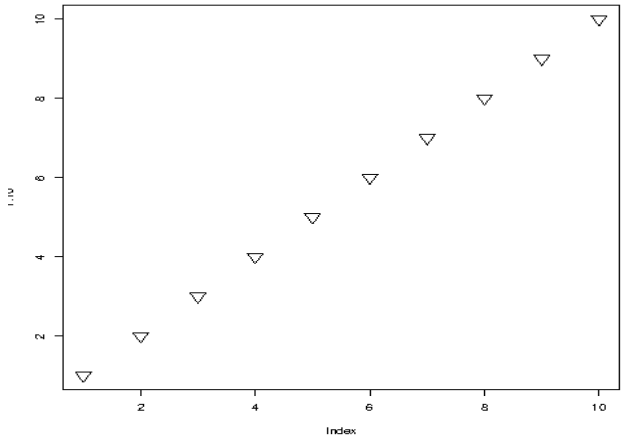


Рис. 13.1. Таблиця 25-ти стандартних маркерів і відповідних їм чисельних кодів

Наприклад, при pch = 25 символи перетворяться у зафарбовані трикутники:

```
plot(1:10, pch=25, cex=2)
```



2. Розмір маркера

Розмір маркерів задається за допомогою аргументу **cex** (character extension – розмір символу), який за замовчуванням дорівнює 1. Зменшення або збільшення цього параметра призводить до відповідних пропорційним змін розміру символів. При необхідності ми можемо також змінити ширину лінії обведення символу. Для цього служить параметр **lwd** (line width – ширина лінії).

3. Колір маркера

Колір будь-якого графічного об'єкта може бути заданий декількома способами:

° за назвою кольору: наприклад, col = "red" (червоний), col = "green" (зелений), або col = "black" (чорний).

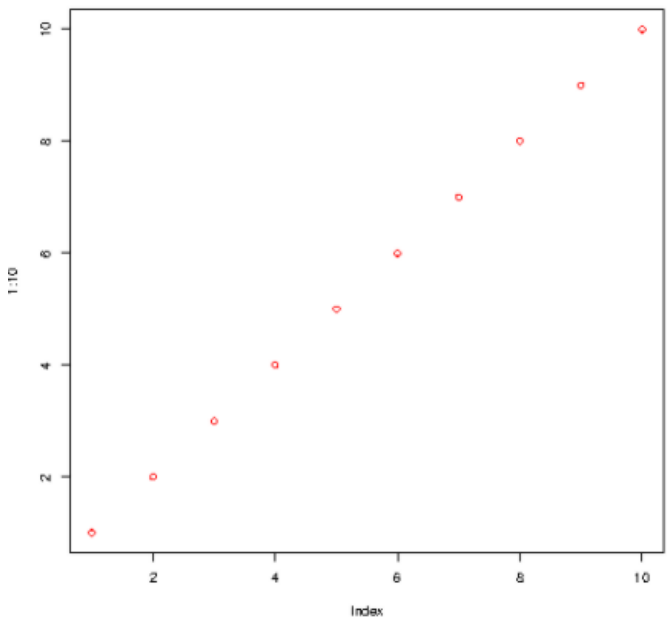
Всього в R є 675 стандартних кольорів. Їх назви доступні з командою colors ();

° шляхом безпосередньої вказівки червоного, зеленого і синього компонентів RGB спектра, наприклад: "#RRGGBB";

° за чисельним кодом, наприклад: col = 2 (червоний), col = 3 (зелений), або col = 1 (чорний).

Колір маркерів задається за допомогою аргументу **col** (color – колір).

```
plot(1:10, col="red")
```



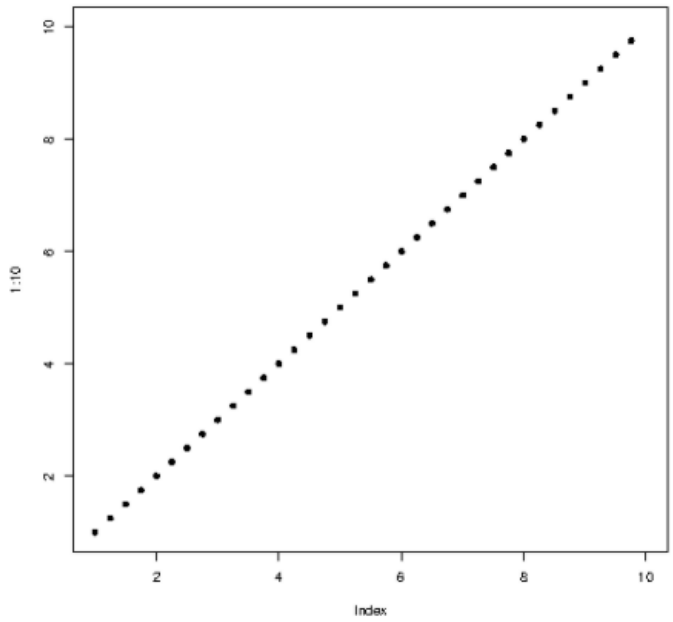
Щоб побудувати лінію, використовується функція plot() та параметр **type** зі значенням "l".

4. Ширина лінії

Ширина лінії задається за допомогою аргументу **lwd** (від line width) функції plot (). Щоб змінити ширину лінії, використовується параметр **lwd** (1 за замовчуванням, тоді як 0.5 означає на 50% менше і 2 означає на 100% більше).

За замовчуванням лінія суцільна. Щоб вказати формат рядка, використовується параметр **lty** зі значенням від 0 до 6. Наприклад, якщо lty=3 замість суцільної буде показано пунктирну лінію:

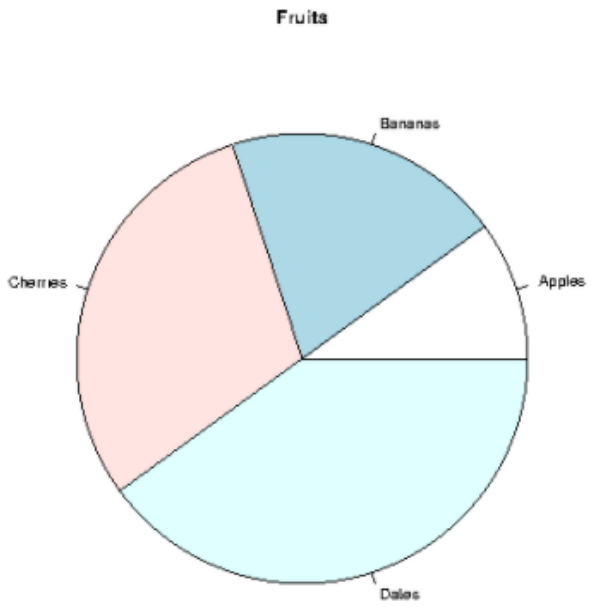
```
plot(1:10, type="l", lwd=5, lty=3)
```



13.3. Типи графіків в R

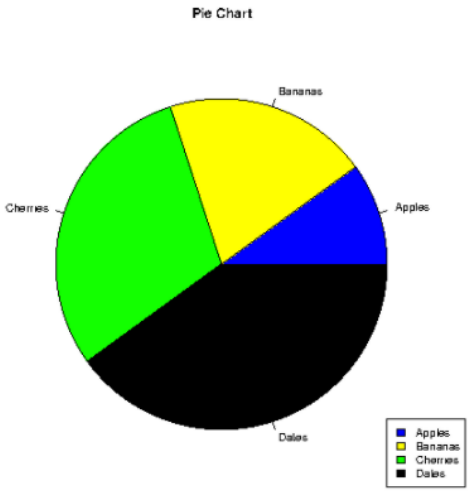
Для побудови кругових діаграм використовується функція **pie()**. Щоб додати мітку до діаграми, використовується параметр **label**, а за допомогою параметра **main** додається заголовок.

```
# Create a vector of pies
x <- c(10,20,30,40)

# Create a vector of labels
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")

# Display the pie chart with labels
pie(x, label = mylabel, main = "Fruits")
```



Щоб додати список пояснень для кожного сектору, використовується функція **legend()**.

```
# Create a vector of labels
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")

# Create a vector of colors
colors <- c("blue", "yellow", "green", "black")

# Display the pie chart with colors
pie(x, label = mylabel, main = "Pie Chart", col = colors)

# Display the explanation box
legend("bottomright", mylabel, fill = colors)
```



Гістограма дозволяє наочно представити розподіл значень аналізованої змінної. В системі R для побудови гістограм використовується функція **hist ()**. Її основним аргументом виступає ім'я аналізованої змінної.

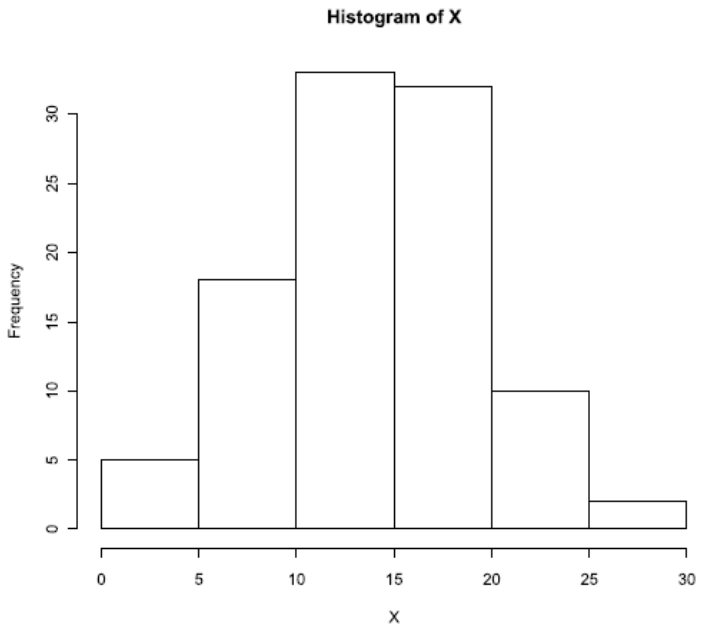
Як приклад створимо нормально розподілену сукупність X з 100 спостережень із середнім значенням 15 і стандартним відхиленням 5:

X <- rnorm (n = 100, mean = 15, sd = 5)

Для створення змінної X використана функція **rnorm ()** (від random – випадковий, і norm – нормальний). Використовуючи генератор випадкових чисел, ця функція формує нормально розподілені сукупності з заданими розміром (n), середнім значенням (mean) і стандартним відхиленням (sd).

Зобразити значення змінної X у вигляді гістограми дуже просто:

hist (X)



Функція hist () автоматично вибирає кількість стовпців для відображення на графіку, створює назви осей і заголовок графіка. Такого малюнка, одержуваного з використанням автоматичних налаштувань, може виявитися цілком достатньо (наприклад, при проведенні швидкого розвідувального аналізу даних). Однак часто потрібно його додаткове доопрацювання. Перш за все, важливо звернути увагу на розмір кроку для розбиття даних на класи при побудові гістограми. У наведеному вище прикладі програма автоматично розбила значення змінної X на 6 класів. Однак таке грубе розбиття може недостатньо точно відображати властивості аналізованої сукупності. Для більш детального вивчення цих властивостей можна збільшити ділення даних на класи (використовувати

менший класовий проміжок). Зробити це дозволяє аргумент breaks функції hist (). За потреби стовпці гістограми можна залити бажаним кольором. Для цього слід скористатися аргументом col функції plot (). У наведеному прикладі вибрано світло-блакитний колір стовпців ("lightblue"):

hist (X, breaks = 20, col = "lightblue")

Як бачимо, результатом виконання попередньої команди стала гістограма з двадцятьма стовпцями, що дозволяє більш детально проаналізувати розподіл значень змінної X.

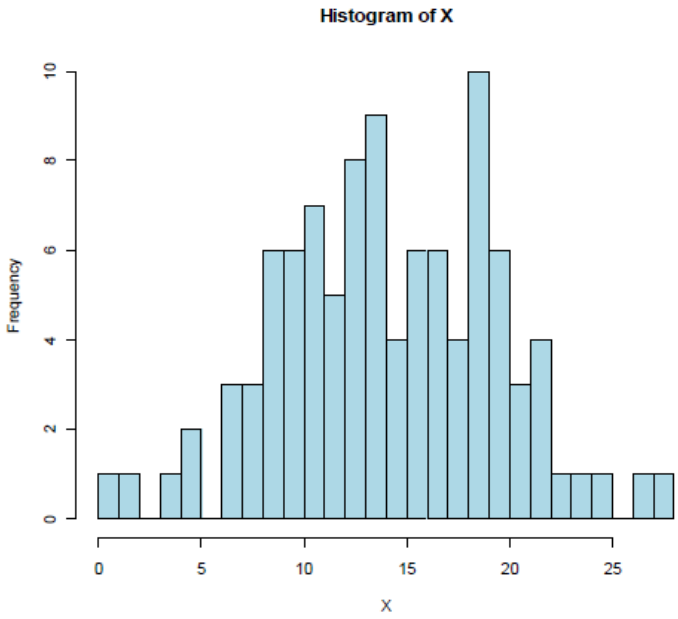
Histogram of X

X

Frequency

0 5 10 15 20 25

0 2 4 6 8 10



За замовчуванням функція hist () відображає по осі ординат частоти для кожного класу значень X. Таку поведінку функції можна змінити, надавши аргументу freq (від frequency – частота) значення FALSE. У цьому випадку вісь ординат буде відображати щільність ймовірності кожного класу так, що сумарна площа під гістограмою складе 1:

hist (X, breaks = 20, freq = FALSE)

У ряді випадків, зокрема при невеликому числі спостережень, гістограми можуть давати неправильне уявлення про властивості сукупності, наприклад, через невелику числа рідко розташованих стовпців:

X <- rnorm (n = 50, mean = 15, sd = 5)

hist (X, breaks = 20, freq = FALSE, col = "lightblue").

Замість гістограми (або паралельно з нею) в таких випадках рекомендується скористатися кривою щільності ймовірності. Оцінка щільності ймовірності виконується за допомогою функції density (), яку можна застосувати в якості аргументу функції plot () для графічного зображення результату:

plot (density (X))

Гладкість одержуваної кривої регулюється за допомогою аргументу bw (від bandwidth – ширина вікна), наприклад:

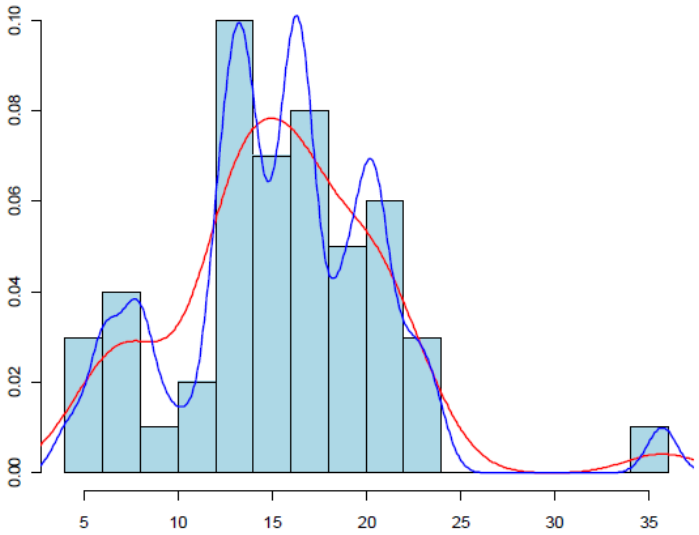
plot (density (X, bw = 0.8)).

Для повноти картини гістограму можна поєднати з кривою щільності ймовірності. При цьому спочатку необхідно побудувати саму гістограму, а потім додати до неї криву щільності за допомогою функції lines ():

hist (X, breaks = 20, freq = FALSE, col = "lightblue", xlab = "", ylab = "")

lines (density (X), col = "red", lwd = 2)

lines (density (X, bw = 0.8), col = "blue", lwd = 2)



У якості керуючих аргументів функції lines () були застосовані аргументи col (для установки кольору лінії) і lwd (для установки товщини лінії).

Наведені приклади показують універсальність ряду інших аргументів, які керують поведінкою plot (), hist () та інших графічних функцій R високого рівня.

Висновок до лекції 13

Найбільш використовуваними графічними функціями високого рівня в R є plot (), hist (), boxplot (), scatterplot () і pairs (), з використанням функцій низького рівня до побудованого графіку можуть бути додані додаткові елементи (текст, лінії, легенда тощо). Особливостями деталей графічного зображення управляє набір параметрів, загальний для більшості функцій, який визначає колір, тип і розмірів символів або маркерів, товщину і характер ліній, штрихування, рамку графіка тощо.

Функція plot () використовується для побудови графіків в R. Поведінка цієї функції визначається класом об'єктів, які вказуються в якості її аргументів. За допомогою plot () можна створити великий набір різнотипних графіків. Функція plot () має велику кількість управляючих параметрів, які дозволяють здійснити налаштування зовнішнього вигляду графіка.

Питання для закріплення

1. Для чого використовується функція plot ()? Які її параметри?
2. Назвіть загальні параметри - аргументи графічних функцій.
3. Які типи графіків в R ви знаєте?

Список рекомендованої літератури

1. R Graphics // Електронний ресурс. Режим доступу: https://www.w3schools.com/r/r_graph_plot.asp
2. Data visualization // Електронний ресурс. Режим доступу: https://r4ds.had.co.nz/data-visualisation.htm
3. Plotly R Open Source Graphing Library // Електронний ресурс. Режим доступу: https://plotly.com/r/
4. Producing Simple Graphs with R // Електронний ресурс. Режим доступу: https://sites.harding.edu/fmccown/r/

Лекція 14.
Архітектурні моделі Big Data. Технології віртуалізації. Гіпервізори. Контейнерна технологія виконання програмного коду на сервері. SaaS, PaaS і IaaS

План лекції

14.1. Архітектурні моделі інженерії Big Data.
14.2. Центри обробки даних та хмарні обчислення.
14.3. Технології віртуалізації.
14.4. Шари абстракції.
14.5. Гіпервізори.
14.6. Контейнерна технологія виконання програмного коду на сервері.
14.7. Інжиніринг даних.

14.1. Архітектурні моделі інженерії Big Data

Перетворення даних у цінну інформацію потребує обчислювальної потужності та пам'яті. Різні архітектури IoT мають різні підходи щодо того, де і коли дані обробляються та зберігаються (рис. 14.1).

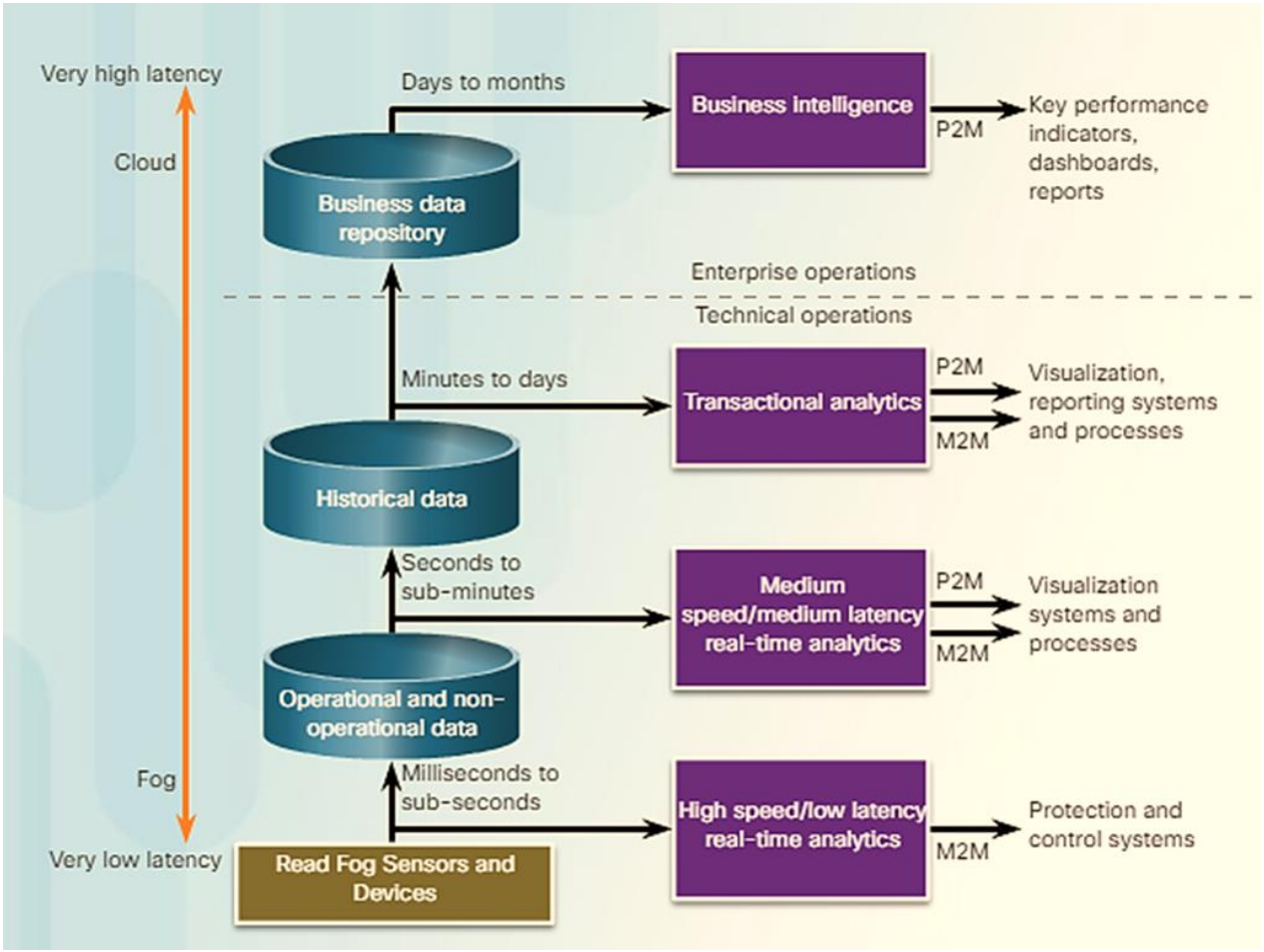


Рис. 14.1. Аархітектурна модель Device-Network-Cloud [1]

Наприклад, в архітектурній моделі Device-Network-Cloud усі точки даних, зібрані датчиками, що входять до підключеного пристрою, надсилаються безпосередньо в хмару для зберігання та оброблення.

Наприклад, дані, зібрані пристроєм для відстеження фітнес-занять, передаються до хмари. Там вони трансформуються за допомогою описової аналітики і подаються користувачеві у вебпрофілі. Ця архітектурна модель проста, але не масштабована. Коли кількість датчиків збільшується разом із кількістю генерованих точок даних або коли обробка даних вимагає значно коротшого часу відгуку, це ситуації, коли дані потрібно обробляти ближче до місця їх генерування. Тут використовується архітектура Device-Gateway-Network-Cloud. Залежно від програми дані можуть бути оброблені майже відразу після їх генерування, поблизу від джерела їх створення на шлюзі або інших проміжних місцях у мережі (туманні обчислення). Ця область джерела також відома як край, і з цієї причини такий підхід також називають крайовою аналітикою. Прикладами програм, які потребують обчислення туману, є мережі датчиків, які географічно розподілені, наприклад, датчики вологості землі на винограднику та датчики руху на кожному перехресті у місті. Туманні обчислення допомагають скоротити час відгуку (низька затримка) та зменшити кількість даних, які необхідно надсилати до хмари. Наприклад, обчислення туману видаляє надлишкові дані, коли змінна датчика не змінюється, оскільки не раціонально продовжувати передавати одні і ті ж значення. Натомість дані передаються лише тоді, коли їх значення змінюються.

Незалежно від використовуваної архітектури IoT, більшість або усі дані з часом будуть збиратися та зберігатися в хмарі, де є обчислювальна потужність та ємність для зберігання. Мережі центрів обробки даних використовують технологію віртуалізації. У центрах обробки даних доступні тисячі (або сотні тисяч) серверів, а також накопичувач потужності для зберігання даних через високошвидкісні мережеві з'єднання. Технологія віртуалізації дозволяє створювати всередині кожного фізичного сервера одну або кілька віртуальних

машин, де може працювати процес аналізу даних. Провайдери хмарних платформ мають мережу центрів передачі даних про несправності.

Дані рухаються по мережевій інфраструктурі. Організації залежать від своїх IT-операцій. Здатність інфраструктури швидко робити доступними ресурси безпосередньо впливає на швидкість передачі даних. Використання великих даних для отримання інформації та розуміння бізнесу вимагає потужних рішень, таких як центри обробки даних (ЦОД). Наприклад, по мірі розвитку організацій вони потребують збільшення кількості обчислювальної потужності та місця на жорсткому диску. Якщо залишити його без розгляду, це негативно вплине на здатність організації надавати життєво важливі послуги. Втрата життєво важливих послуг означає нижчу задоволеність клієнтів, менший дохід, а в деяких ситуаціях і втрату майна.

Великі підприємства, як правило, володіють ЦОД для управління потребами організації сховища та доступу до даних. У ЦОД один орендар підприємства є єдиним замовником, що користується послугами ЦОД. Однак, оскільки обсяг даних продовжує розширюватися, навіть великі підприємства розширюють свої можливості зберігання даних, використовуючи послуги ЦОД, які можуть використовуватися для задоволення внутрішніх потреб IT (приватна хмара). ЦОД можуть також пропонувати ці самі товари та послуги іншим компаніям та організаціям (публічна хмара).

14.2. Центри обробки даних та хмарні обчислення

Щоб допомогти вирішити чотири Vs Big Data (обсяг, різноманітність, швидкість та правдивість), багато організацій звертаються до хмарних обчислень. Хмарні обчислення підтримують різноманітні проблеми управління даними:

- доступ до даних організації будь-де та в будь-який час;
- упорядкування IT-операцій організації, підписка лише на потрібні послуги;

• зменшення потреб в IT-обладнанні, технічному обслуговуванні та управлінні на місці;

• зменшення витрат на обладнання, енергію, фізичні потреби в установках та потреби в навчанні персоналу;

• швидке реагування на збільшення потреб у обсязі даних;

• витрати на обчислення та зберігання, а не інвестування в інфраструктуру, капітальні витрати перетворюються на операційні видатки.

Три основні послуги хмарних обчислень, визначені Національним інститутом стандартів та технологій (National Institute of Standards and Technology, NIST) у спеціальній публікації 800-145:

• **SaaS** – програмне забезпечення як послуга (Software as a Service);

• **PaaS** – платформа як послуга (Platform as a Service);

• **IaaS** – інфраструктура як послуга (Infrastructure as a Service).

Провайдери хмарних послуг розширили цю модель, щоб забезпечити IT-підтримку для кожної з хмарних обчислювальних служб (ITaaS).

Наразі у світі існує понад 3000 центрів обробки даних, які пропонують загальні послуги розміщення і оброблення даних для організацій. Існує набагато більше центрів обробки даних, які належать приватній галузі та управляються ними для власного використання.

Дані центри – це централізовані місця, що містять велику кількість обчислювальної та мережевої техніки. Це обладнання використовується для збору, зберігання, обробки, розповсюдження та надання доступу до величезної кількості даних. Основна його функція – забезпечити безперервність бізнесу, зберігаючи обчислювальні послуги доступними, коли і де вони потрібні.

Практично кожна організація потребує власного ЦОД або доступу до ЦОД. Деякі організації будують і підтримують власні центри обробки даних. Інші організації орендують сервери у місцях спільного розташування. Є й інші, які використовують публічні, хмарні сервіси. Веб-сервіси Amazon, Microsoft Azure, Rackspace та Google – приклади компаній, які надають публічні хмарні послуги.

Через операційну складність ЦОД дуже мало організацій управляють власним ЦОД. Тому багато організацій орендують простір у спеціалізованих центрах даних, що належать постачальникам послуг, для розміщення їх систем.

14.3. Технології віртуалізації

Операційні системи (ОС) відокремлюють програми від апаратних засобів. ОС створюють "абстракцію" деталей апаратних ресурсів програми. Віртуалізація відокремлює ОС від апаратного забезпечення.

Хмарні провайдери пропонують послуги, які можуть динамічно надавати сервери за потребою. Віртуалізація сервера використовує переваги простоюючих ресурсів на фізичній машині та консолідує кілька віртуальних серверів на одній машині. Це також дозволяє на одній апаратній платформі використовувати декілька операційних систем. Наприклад, на рис.14.2 оригінальні вісім виділених серверів були об'єднані у два сервери за допомогою гіпервізорів для підтримки декількох віртуальних екземплярів ОС.

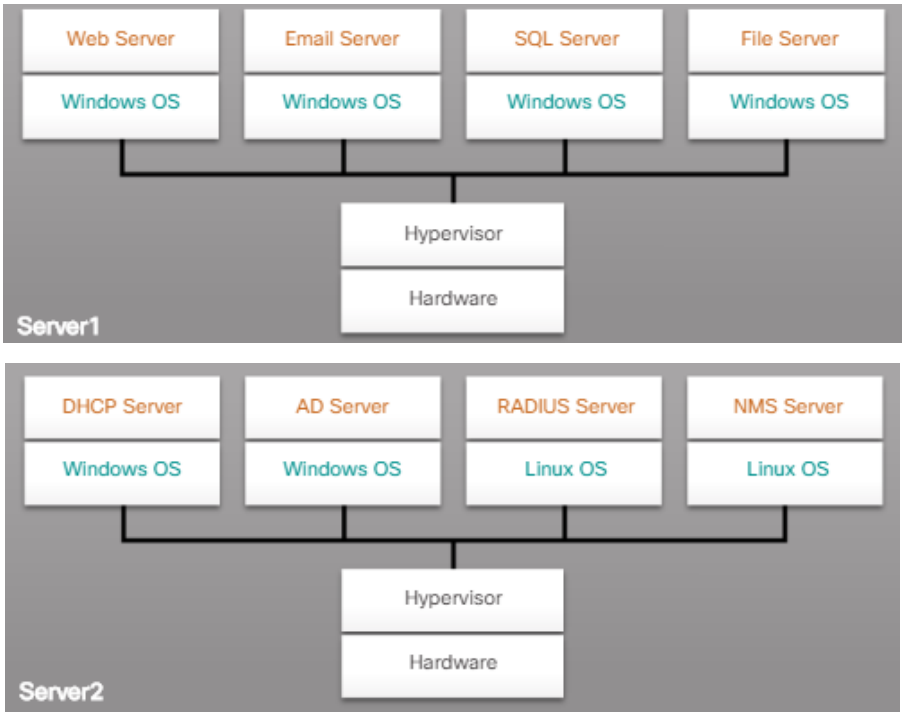


Рис. 14.2. Встановлення гіпервізора ОС [1]

Гіпервізор – це програма, прошивка або апаратне забезпечення, яке додає шар абстракції поверх реального фізичного обладнання.

Шар абстракції використовується для створення віртуальних машин, які мають доступ до всіх апаратних засобів фізичної машини, таких як процесори, пам'ять, дискові контролери та NIC. Кожна з цих віртуальних машин працює з окремою операційною системою. Завдяки віртуалізації підприємства можуть консолідувати кількість серверів, якими вони володіють та працюють. Наприклад, не рідкість 100 фізичних серверів консолідуватись як віртуальні машини на вершині 10 фізичних серверів за допомогою гіпервізорів.

Використання віртуалізації зазвичай включає надмірність для захисту від єдиної точки відмови. Надлишок може бути реалізований різними способами. Якщо гіпервізор не працює, VM можна перезапустити на іншому гіпервізорі. Крім того, однакова VM може працювати одночасно на двох гіпервізорах, копіюючи між собою інструкції оперативної пам'яті та процесора. Якщо один гіпервізор виходить з ладу, VM продовжує працювати на іншому гіпервізорі.

14.4. Шари абстракції

Щоб пояснити, як працює віртуалізація, корисно використовувати шари абстракції в комп'ютерних архітектурах. Шари абстракції (програми, ОС, обладнання) також використовуються еталонною моделлю OSI, щоб допомогти описати мережеві протоколи (рис.14.3).

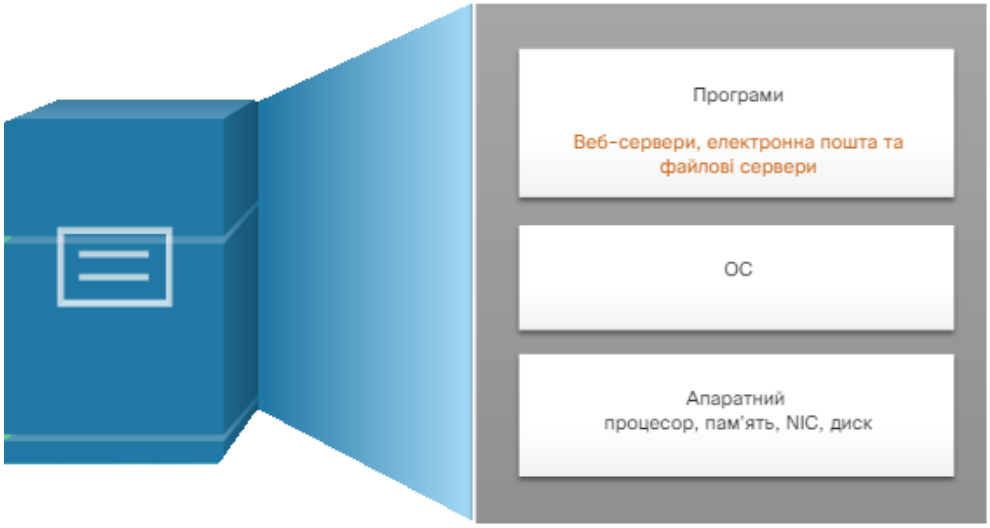


Рис. 14.3. Шари абстракції комп'ютерної системи (програми, ОС, обладнання) [1]

На кожному з цих шарів абстракції деякий тип коду програмування використовується як інтерфейс між шаром внизу та шаром вгорі. Наприклад, мова програмування С часто використовується для програмування мікропрограмного забезпечення, яке здійснює доступ до обладнання.

Приклад віртуалізації показаний на рис.14.4. Гіпервізор встановлюється між прошивкою та ОС. Гіпервізор може підтримувати кілька примірників ОС.

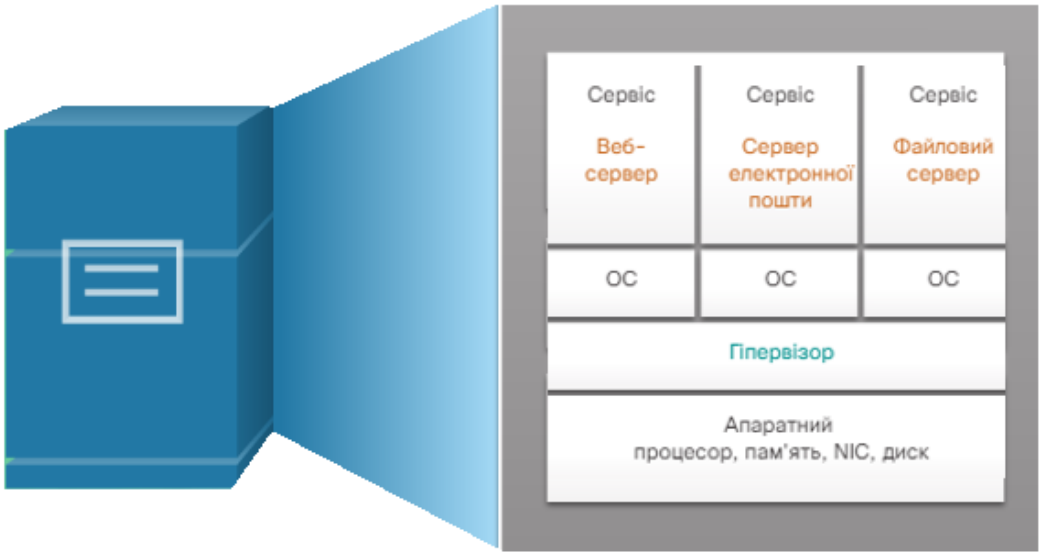


Рис. 14.3. Шари абстракції віртуальної архітектури [1]

14. 5. Гіпервізори

Гіпервізор – це програмне забезпечення, яке створює та запускає екземпляри VM. Комп'ютер, на якому гіпервізор підтримує одну або декілька VMs, є хост-машиною. Існує два наступні типи гіпервізорів.

• **Гіпервізор типу 1** – є підходом «голого металу», оскільки гіпервізор встановлюється безпосередньо на апаратному забезпеченні, (рис.14.4). Гіпервізори типу 1 зазвичай використовуються на серверах підприємства.

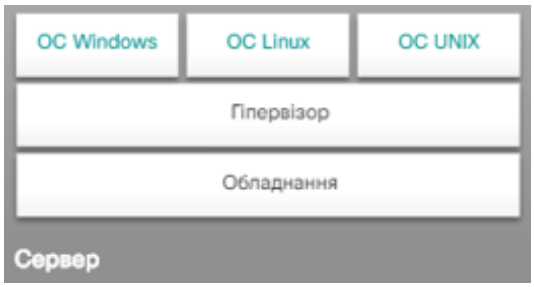


Рис. 14.4. Гіпервізор типу 1 [1]

• **Гіпервізор 2 типу** – є підходом «розміщення». Гіпервізор типу 2 додає додатковий шар абстракції. Це відбувається тому, що гіпервізор – це програма, що працює на ОС фізичного хоста, а додаткові екземпляри ОС встановлюються в гіпервізорі (рис. 14.5).

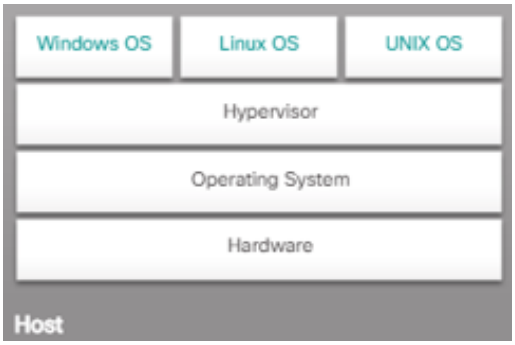


Рис. 14.5. Гіпервізор типу 2 [1]

14.6. Контейнерна технологія виконання програмного коду на сервері

Гіпервізори дозволяють кожній віртуальній машині мати власну операційну систему під час спільного використання одного і того ж обладнання. Ця конфігурація є марною, якщо операційні системи, що використовуються у віртуальних машинах, такі самі, як операційна система, що працює на хост-комп'ютері. Контейнери вирішують цю проблему.

Контейнер – це спеціалізована «віртуальна область», де програми можуть працювати незалежно одна від одної під час спільного використання однієї ОС та обладнання. З точки зору програми, це єдиний додаток, що працює на комп’ютері. Обмінюючись операційною системою хосту, більшість програмних ресурсів повторно використовуються, що оптимізує роботу.

Контейнеру потрібна лише необхідна частина операційної системи, системні ресурси та будь-які програми та бібліотеки, необхідні для запуску програми. Це дозволяє серверу підтримувати набагато більше контейнерів, ніж це могла зробити віртуальна машина в будь-який момент (рис. 14.6).

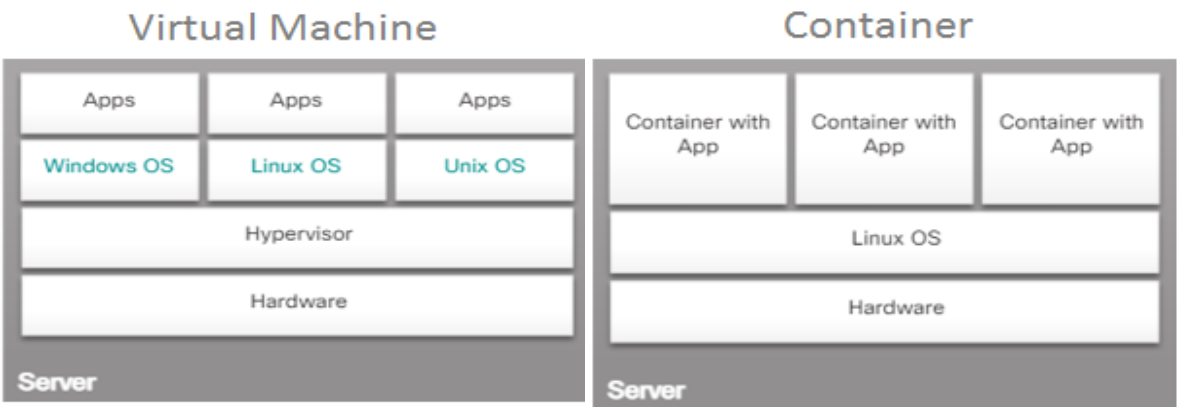


Рис. 14.6. Структура контейнеру порівняно з традиційним гіпервізором [1]

Більше програм можна запустити на сервері за допомогою контейнерної технології, контейнери вимагають, щоб операційна система віртуальної машини була такою ж, як і хост-комп'ютер. Якщо є потреба в декількох операційних системах, слід використовувати гіпервізори.

Центри обробки даних також можуть використовувати віртуалізацію для зниження витрат і розширення пропозицій хмарних постачальників.

Amazon Web Services (AWS) – хмарний постачальник послуг, який пропонує обчислювальні ресурси та послуги за потребою в хмарі. Це означає, що можна працювати на вимогу на одному або декількох віртуальних серверах на AWS, коли вони потрібні протягом певного часу.

За допомогою AWS можна зберігати дані, розміщувати веб-сайти та веб-додатки, розміщувати систему управління навчанням (Learning Management System, LMS) та обробляти великі дані, створені IoT. Це приклад IaaS (Infrastructure as a Service), де створення обчислювальної інфраструктури перетворюється на придбання послуги. Машинне навчання Amazon дозволяє розробникам створювати програми для виявлення шахрайства, прогнозування попиту, цільового маркетингу та прогнозування кліків. Алгоритми машинного навчання Amazon створюють моделі машинного навчання (ML), знаходячи шаблони в існуючих даних. Потім ці моделі обробляють нові дані та генерують прогнози. Одним із можливих застосувань моделі ML є передбачення, з якою ймовірністю клієнт придбає певний товар, виходячи з його минулої поведінки. Існує багато хмарних провайдерів. У Північній Америці вони включають

Microsoft Azure та Google Cloud. У Європі деякі хмарні постачальники – Aruba Cloud, UpCloud та CenturyLink.

Віртуалізація пам'яті поєднує в собі фізичне зберігання з декількох мережевих пристроїв зберігання даних, що видається одним пристроєм зберігання даних. Пристрій зберігання даних керується з центральної консолі. Віртуалізація пам'яті робить резервне копіювання, архівування та відновлення простішими та швидшими. Віртуалізація зберігання реалізується за допомогою програмного забезпечення, або з апаратними та програмними гібридними пристроями. Переваги віртуалізації зберігання включають збільшення ємності зберігання, автоматизоване управління, скорочення часу простою та спрощення оновлення.

Віртуальна мережа (NV) – це створення віртуальних мереж у рамках віртуалізованої інфраструктури. Процес поєднує апаратні та програмні мережеві ресурси та мережеві функціональні можливості в єдину, створену на основі програмного забезпечення адміністративну структуру. Ця сутність є віртуальною мережею. Віртуалізація мережі поєднує мережеві ресурси, розділяючи пропускну здатність на канали. Кожен канал не залежить від інших, і призначається певному серверу або пристрою. Кожен канал незалежно захищений. Кожен абонент мережі має спільний доступ до всіх ресурсів цієї мережі. Для віртуалізації мережі функція площини управління знімається з кожного пристрою і виконується централізованим контролером (рис. 14.7).

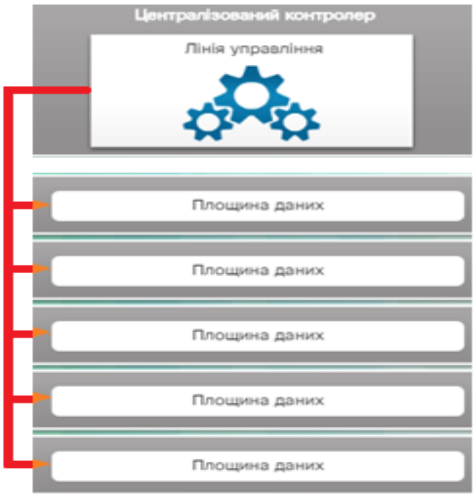


Рис. 14.7. Централізований контролер та лінія управління [1]

Централізований контролер повідомляє функції площини управління кожному пристрою. Кожен пристрій тепер може зосередитися на передачі даних, тоді як централізований контролер управляє потоком даних, підвищує безпеку та забезпечує ланцюжок послуг. Дані центри часто є середовищем, де живуть багато користувачів. NV може надавати окремі віртуальні мережі різним клієнтам у віртуальному середовищі. Ця віртуальна мережа повністю відокремлена від інших мережевих ресурсів, трафік розділяється на зони або контейнери, щоб не змішуватися з іншими ресурсами.

14.7. Інжиніринг даних

Інжиніринг даних включає інформаційну систему, де інформація (дані) збирається або формується, обробляється, зберігається, поширюється та аналізується. Можливість аналізу даних зазвичай виконується за допомогою бази даних та системи управління базами даних (СУБД). Інженерія даних та аналіз даних корисні для будь-якого бізнесу чи організації, яка хоче спрямувати свої ресурси на основі змістовної інформації та статистики.

Реляційна база даних та мова програмування структурованої мови запитів (SQL) є основою системи управління реляційними базами даних (RDMS). SQL – мова програмування, розроблена для збору інформації з реляційних баз даних за допомогою систем управління реляційними базами даних (RDBMS). Реляційні системи баз даних, такі як MySQL, Microsoft SQL Server, Oracle та IBM DB2, є найпопулярнішими системами управління базами даних. У RDBMS може бути кілька користувачів з багатьма транзакціями баз даних. Модель транзакцій Atomic, Consistent, Isolated and Durable (ACID) визначає, як транзакції бази даних підтримують цілісність даних та переживають збої.

На початку 2000-х, поява Web 2.0, електронної комерції та таких компаній, як Google, дали зрозуміти, що реляційні бази даних не змогли задовольнити об'єм та швидкість пошукових запитів у мережі. Для задоволення цього попиту Google розробила розподілену файлову систему Google (GFS), алгоритм розподіленої паралельної обробки MapReduce та розподілену базу даних NoSQL BigTable. У

2004 році Джеффрі Дін та Санджай Гемамат з Google опублікували статтю "Спрощена обробка даних на великих кластерах", яка надихнула двох програмістів Дуга Кейтінга та Майка Кафарелла створити Apache Hadoop. Після цього підхід MapReduce став основою для розробки екосистеми Hadoop і бази даних HBase, а також бази даних NoSQL з ключовими значеннями Amazon Dynamo.

Бази даних NoSQL можуть використовувати підхід для зберігання ключових значень замість підходу на основі реляційної таблиці. Інші бази даних NoSQL зберігають дані як структуровані документи у форматах XML або JSON. Бази даних NoSQL значно швидші, ніж реляційні, і можуть імпортувати неструктуровані дані. Бази даних NoSQL розроблені для масштабування по горизонталі, а це означає, що ємність зберігання та управління може бути збільшена просто додаванням інших машин до кластеру. До найпопулярніших систем NoSQL належать MongoDB, Couchbase, Riak, Memcached, Redis, CouchDB, Hazelcast, Apache Cassandra, HBase та Dynamo, які є всіма програмними продуктами з відкритим кодом.

Усі ці технології стали вирішенням проблеми Big Data. Дані настільки великі, швидкі або різноманітні, що ним неможливо керувати одним комп'ютером. З цієї причини прийнято називати ці програмні рішення «технологіями великих даних». Насправді проблеми з великими даними не можуть бути зведені до жодної технології, але повинні включати нові та старі технології.

З появою IoT та Big Data з’являються нові категорії робочих місць та коригування існуючих робочих місць.

Оцифровка бізнесу створює багато доступних бізнес-даних. Щоб отримати необхідні дані, важливо мати можливість задати правильне питання правильним чином. Бізнес-аналітик – це людина, яка може вивчати бізнес чи галузь, а потім сформулювати конкретне питання. Бізнес-аналітики – це експерти з даних, які співпрацюють із зацікавленими сторонами компанії, щоб визначити проблемне питання. Це питання потім переформулюється в конкретну проблему даних.

Потім бізнес-аналітики створюють звіти про бізнес-аналітику різних типів для зацікавлених сторін компанії.

Аналітики даних запитують та обробляють дані, надають звіти, узагальнюють та візуалізують дані. Вони використовують існуючі інструменти та методи для вирішення проблеми, допомагають розуміти конкретні запити за допомогою спеціальних звітів та графіків. Аналітикам даних необхідно зрозуміти основні статистичні принципи, процес очищення різних типів даних, візуалізацію даних та дослідницький аналіз даних. Деякі інструменти та програми, які допомагають аналітикам даних виконувати свою роботу, – це SAS, Rapid Miner та мови програмування, такі як R або Python.

Аналітик даних бере вихідні дані і перетворює їх на змістовну інформацію. Такі дослідники застосовують статистику, машинне навчання та аналітичні підходи для відповіді на критичні питання бізнесу. Наука даних – це вже існуюче поле, яке розширилося завдяки IoT та Big Data.

Аналітики даних повинні інтерпретувати та надавати результати своїх висновків методами візуалізації, будуючи програми для наукових даних. Вони працюють з наборами даних різного розміру та форм та запускають алгоритми на великих наборах даних. Деякі інструменти, які допомагають вченим даним виконувати свою роботу, – це Python, R, Scala, Apache Spark, Hadoop, data mining tools and algorithms, machine learning, statistics.

Жодна з трьох деталей, описаних вище, не може існувати без інженера даних. Інженери даних створюють інфраструктуру, яка підтримує Big Data. Вони проектують та будують платформу, на якій усі ці дані зберігаються та обробляються. Інженери даних також керують усіма цими даними. Вони забезпечують доступність даних для науковців та аналітиків.

Інженери даних можуть інтегрувати дані з різних джерел та проводити очищення даних. Однак, оскільки інженери даних розробляють в першу чергу інфраструктуру Big Data для своєї компанії, вони, як правило, не повинні знати машинного навчання чи аналітики. Деякі інструменти та програми, якими інженери даних регулярно користуються, є Hadoop, MapReduce, Hive, Pig,

MySQL, MongoDB, Cassandra, потокова передача даних, NoSQL, SQL та програмування. У деяких середовищах можливе перекриття між аналітиками даних, науковцями та інженерами даних. Коли IoT зростає і великі дані стають ще більш поширеними, посадові інструкції можуть також змінюватися.

Висновок до лекції 14

Віртуалізований центр обробки даних підтримує Big Data та аналітику. За допомогою туманних обчислень дані можуть оброблятися майже відразу після їх створення. Центри обробки даних – це централізовані місця, що містять велику кількість обчислювального та мережевого обладнання. Віртуалізація відокремлює ОС від апаратного забезпечення. Віртуалізація сховища поєднує фізичне сховище з декількох мережевих пристроїв зберігання даних у сховище, що здається єдиним запам'ятовуючим пристроєм.

Мережева віртуалізація (NV) – це створення віртуальних мереж у межах віртуалізованої інфраструктури. Інженерія даних включає в себе збір, оброблення, зберігання, розподіл та аналіз інформації.

Питання для закріплення

1. Які архітектурні моделі інженерії Big Data ви знаєте?
2. Для чого використовуються центри обробки даних та хмарні обчислення?
3. Для чого використовуються технології віртуалізації?
4. Що таке гіпервізори?
5. Що таке контейнери?
6. У чому полягає контейнерна технологія виконання програмного коду на сервері?
7. Що таке інжиніринг даних?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

2. Virtualization Technology // Електронний ресурс. Режим доступу: https://www.sciencedirect.com/topics/computer-science/virtualization-technology

3. What is virtualization technology // Електронний ресурс. Режим доступу: https://pandorafms.com/blog/virtualization-technology/

4. Containerization // Електронний ресурс. Режим доступу: https://www.ibm.com/cloud/learn/containerization

5. What are containers (container-based virtualization or containerization) // Електронний ресурс. Режим доступу: https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization

Лекція 15.
Технології Hadoop Big Data.
Розподілена обробка MapReduce. HDFS

План лекції

15.1. Масштабованість за допомогою великих даних.
15.2. Зберігання та оброблення даних в розподілених файлових системах.
15.3. Розподілені бази даних.
15.4. Розподілена файлова система Hadoop (HDFS).

15.1. Масштабованість за допомогою великих даних

У контексті Big Data масштабованість означає розробку рішення, яке може відповідати експоненційним потребам зростання таких компаній, як Google та Facebook. Це також стосується будь-якої компанії, організації чи урядового агентства, якому потрібно зберігати та аналізувати неструктуровані та структуровані дані з величезних та різних джерел даних. Масштабованість у цьому контексті означає можливість масштабування як зберігання даних, так і обробки даних. Екосистема Hadoop Big Data дотримується підходу до масштабування, подібного до Google, показаного на рис. 15.1.

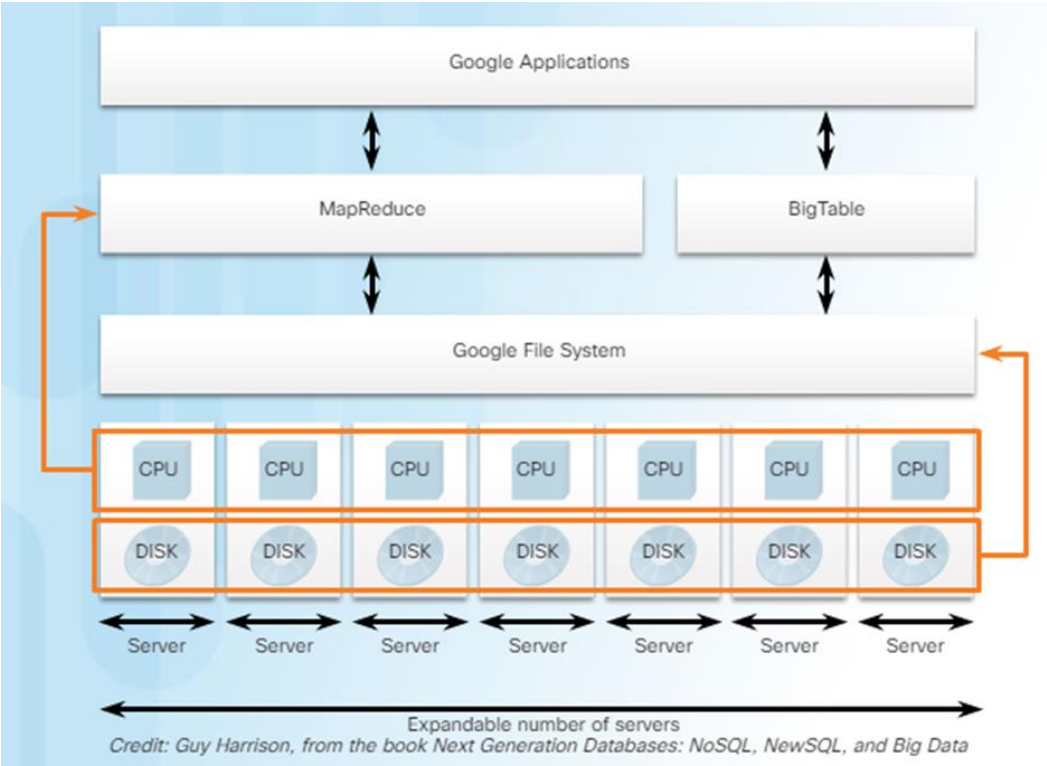


Рис. 15.1. Масштабування зберігання та оброблення даних Google [1]

15.2. Зберігання та оброблення даних в розподілених файлових системах

Hadoop використовує розподілену файлову систему, яка розширюється шляхом додавання більшої кількості комп'ютерів та накопичувачів жорсткого диска. Зберігання даних збільшується за рахунок додавання серверів, які працюють на апаратному забезпеченні. Можливість використання commodity обладнання замість дорогих SAN або NAS-систем зберігання зменшує витрати.

Коли Google потребує додаткової потужності для обробки, використовується модульний центр даних Google – розроблене на замовлення рішення, яке додає додаткові 1000 процесорів, які працюють разом у кластері завдяки технології Google MapReduce. Екосистема Hadoop Big Data також заснована на розподіленій обробці MapReduce. Додавання нових серверів або вузлів до кластеру Hadoop не тільки додає додаткового дискового сховища, але й забезпечує додаткову обробку процесора.

Компаніям, які займаються великими даними або великою кількістю веб-транзакцій, також доводиться вирішувати проблеми масштабування баз даних. Традиційні реляційні бази даних були розроблені для архітектури «клієнт-сервер». Вони не були розроблені для розподілу по декількох серверах баз даних та для роботи з неструктурованими даними або надзвичайно великими об'єктами та рядками даних. Розподілена база даних BigTable від Google, Amazon's Dynamo і Hadoop's HBase – це сховища значень key-value, нереляційні бази даних, розроблені для розподілу на декілька серверів і масштабування в міру додавання нових серверів.

Підтримка доступності є головною проблемою для багатьох компаній, що працюють з Big Data. Веб-сайт, який не зможе відповісти протягом 3 секунд, втратить відвідувачів. Компанії можуть покращити доступність веб-сайтів, розгорнувши веб-сервери, що врівноважують навантаження та завантажують DNS-сервери. Дублікати веб-серверів можна розгортати в центрах обробки даних у різних місцях по всьому світу, щоб покращувати час реакції в Інтернеті.

15.3. Розподілені бази даних

У Big Data доступність позначає швидкість, з якою можна шукати та обробляти великі обсяги даних. Розподілене обчислення, включаючи зберігання та управління базами даних, покращує швидкість оброблення та доступність даних. Реляційна база даних була розроблена для роботи на одному, а не на багатьох серверах.

Для адаптації до потреби в розподілених обчисленнях реляційну базу даних можна розподілити на декількох серверах. Шардінг вимагає великої складності та зменшує реляційну базу даних для доступу на рівні рядків по одному фрагменту. Бізнесу великих компаній, таких як Google і Yahoo, Facebook, Twitter, Amazon потрібно постійно працювати в режимі онлайн та бути доступними 24/7. Екосистеми великих даних, такі як Hadoop допомагають забезпечувати відмовостійкість постачання та оброблення даних.

На початку 2000-х стало зрозуміло, що для каталогізації всіх даних у всесвітній павутині знадобиться нова система управління базами даних. Жодна база даних чи сервер не змогли б обробити велику кількість даних, залучених до такого великого завдання. Для вирішення цього завдання Google потребує розподіленої обчислювальної системи, що складається з кластеру серверів, використовуючи єдину файлову систему, яка поширюється на кілька серверах, де кожен сервер поділяє частину навантаження для обробки.

15.4. Розподілена файлова система Hadoop (HDFS)

Hadoop – це набір утиліт для програмного забезпечення з відкритим вихідним кодом, який полегшує використання мережі багатьох комп'ютерів для вирішення проблем, пов'язаних з великими обсягами даних і обчислень. Hadoop забезпечує програмну основу для розподіленого зберігання та обробки великих даних за допомогою моделі програмування MapReduce.

Усі модулі в Hadoop розроблені з фундаментальним припущенням, що відмови апаратного забезпечення є частими явищами і повинні автоматично оброблятися фреймворком.

Рис. 15.2. Логотип Hadoop [2]

Розподілена файлова система Hadoop (Hadoop Distributed File System, HDFS) – це файлова система, де Hadoop зберігає дані. HDFS не замінює файлову систему Linux на окремих серверах, натомість розташовується поверх кластеру серверів як єдина файлова система, що охоплює весь кластер.

HDFS зберігає дані в 64Мб фрагментах, використовуючи щонайменше три сервери DataNode (рис.15.2). HDFS управляє інформацією через кластер з централізованого сервера координації NameNode.

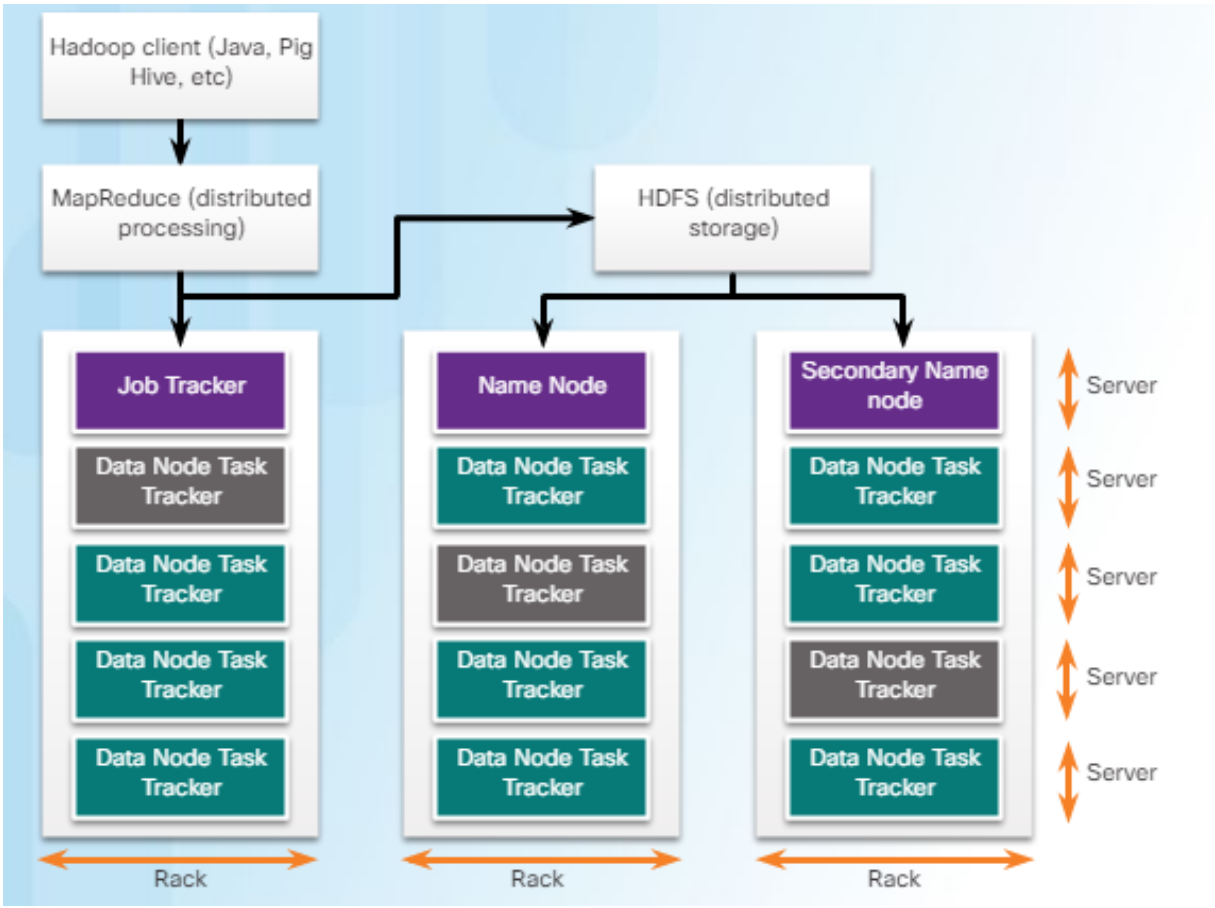


Рис. 15.3. Приклад використання файлової системи Hadoop v 1.0 [1]

Сервер NameNode відстежує, які дані знаходяться на різних серверах DataNode. Коли дані вводяться в систему, вони імпортуються в NameNode, потім NameNode розділяє дані на фрагменти (64Mb), які потім дублюються через три або більше DataNodes залежно від конфігурації. Це забезпечує відмовостійкість, подібну до дзеркального масиву RAID (Redundant Array of Independent Disks), оскільки, якщо одна DataNode має збій, сервер DataNode може бути замінений, а файлова система та дані будуть відновлені з дублікатів DataNodes.

15.5. MapReduce

MapReduce – це розподілений фреймворк для паралелізації оброблення великих наборів даних на великій кількості серверів. MapReduce розділяє обробку даних на дві фази:

1. етап відображення, коли дані розбиваються на частини, які можуть бути оброблені окремими потоками, навіть працюючи на окремих машинах;

2. фаза зменшення, яка поєднує вихід з декількох відображувачів у кінцевий результат (рис. 15.3).

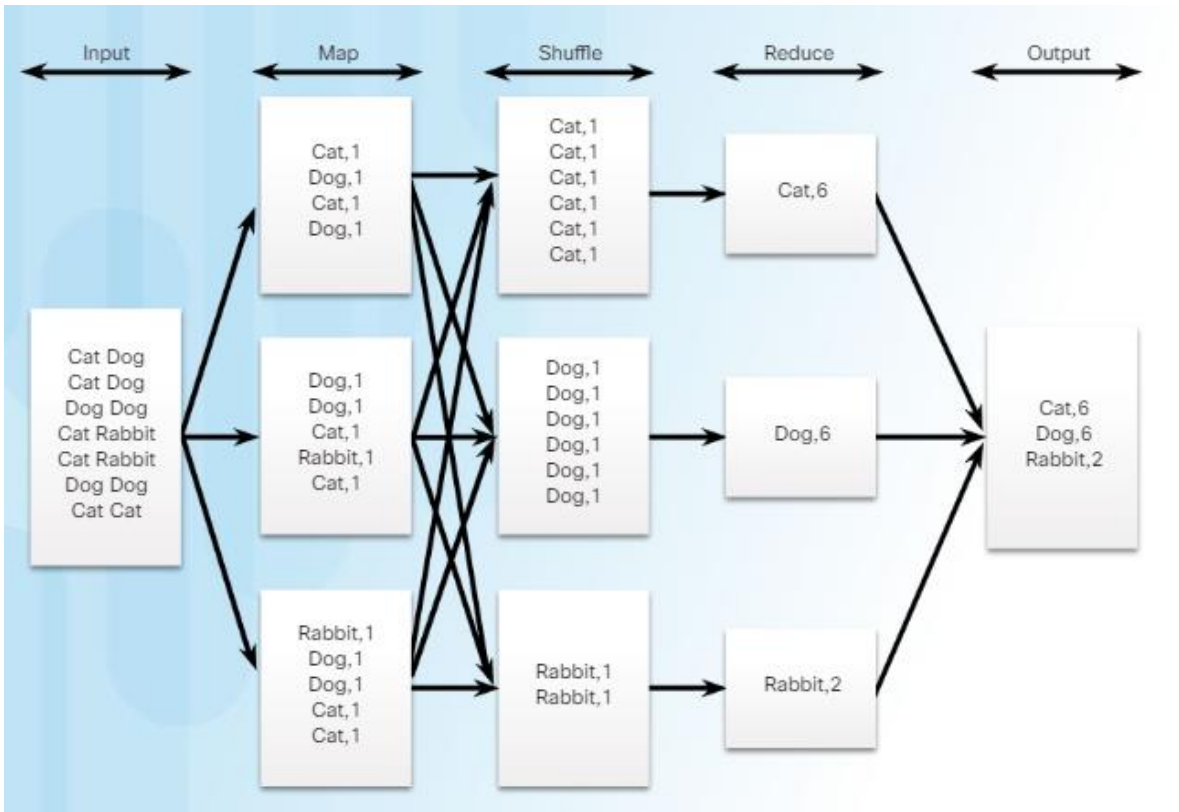


Рис. 15.4. Приклад паралельної обробки даних в MapReduce [1]

Hadoop v2.0 – це екосистема додатків, які працюють разом та включає наступні технології:

- розподілена файлова система HDFS;
- розподілена база даних HBase;
- MapReduce розподіленої обробки;
- Hive забезпечує інтерфейс, подібний SQL.

YARN – платформа для управління обчислювальними ресурсами в кластерах та використання їх для планування програм користувачів, яка узгоджує ресурси для декількох процесорних двигунів:

- **Spark** для запуску процесів у пам'яті;
- **Tez** для запуску пакетних процесів.

Додатковими клієнтськими програмами, які можуть отримати доступ до Hadoop, є **Apache Pig** – платформа високого рівня для створення програм, що працюють на Hadoop та Mahout як інтерфейс машинного навчання.

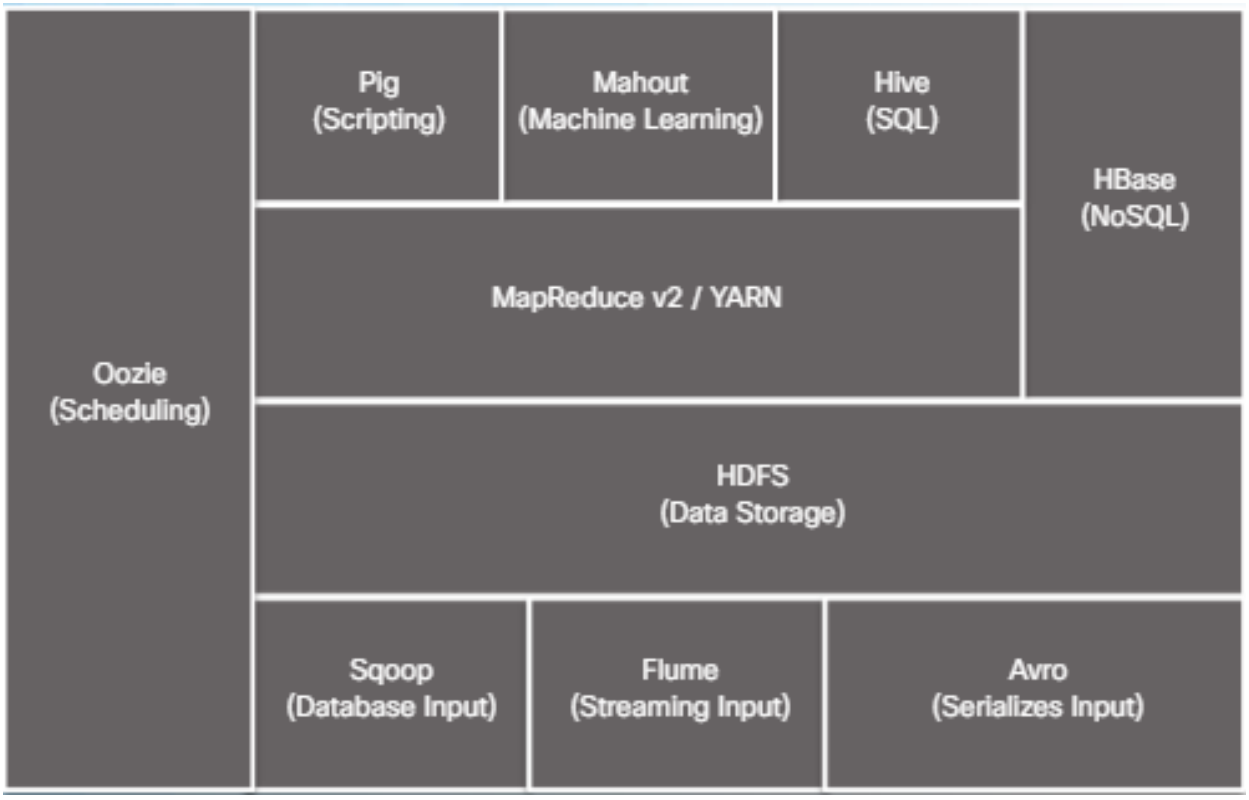


Рис. 15.5. Обробка даних в MapReduce [1]

Висновок до лекції 15

Hadoop – це набір утиліт програмного забезпечення з відкритим вихідним кодом, який полегшує використання мережі багатьох комп'ютерів для вирішення проблем, пов'язаних з великими обсягами даних і обчислень та забезпечує програмну основу для розподіленого зберігання та обробки великих даних.

Розподілена файлова система Hadoop (HDFS) – це файлова система, в якій Hadoop зберігає дані.

MapReduce – це розподілена система обробки і розпаралелювання обчислень великих наборів даних між великою кількістю серверів.

YARN – платформа для управління обчислювальними ресурсами в кластерах та використання їх для планування програм користувачів та розподілу ресурсів для декількох процесорних двигунів.

Apache Pig – платформа високого рівня для створення програм, що працюють на Hadoop та Mahout як інтерфейс машинного навчання.

Питання для закріплення

1. Як відбувається зберігання та оброблення даних в розподілених файлових системах?
2. Яке призначення розподілених баз даних?
3. Яке призначення MapReduce?
4. Що таке Hadoop?
5. Яка структура Hadoop?
6. Що таке YARN?
7. Що таке HDFS?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics
2. Apache Hadoop // Електронний ресурс. Режим доступу: http://hadoop.apache.org/
3. HDFS // Електронний ресурс. Режим доступу: https://www.ibm.com/analytics/hadoop/hdfs
4. MapReduce Tutorial // Електронний ресурс. Режим доступу: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Лекція 16.
Розподілена потокова платформа Kafka.
Переваги Cassandra

План лекції

16.1. Проблема прийому даних.
16.2. Розподілена потокова платформа Kafka.
16.3. Переваги Cassandra.

16.1. Проблема прийому даних

Веб-сервіси, що допомагають надійно і з заданими інтервалами обробляти дані і переміщати їх між різними обчислювальними сервісами, називаються pipeline та виконують приймання, зберігання та оброблення даних (рис.16.1). Для кожного з цих компонентів існує багато програмних платформ, які використовуються для виконання кожного завдання. Залежно від типу даних, типу прийому та вимог до обчислень, компоненти кожного pipeline даних унікальні, часто складаються разом і коригуються для роботи один з одним.

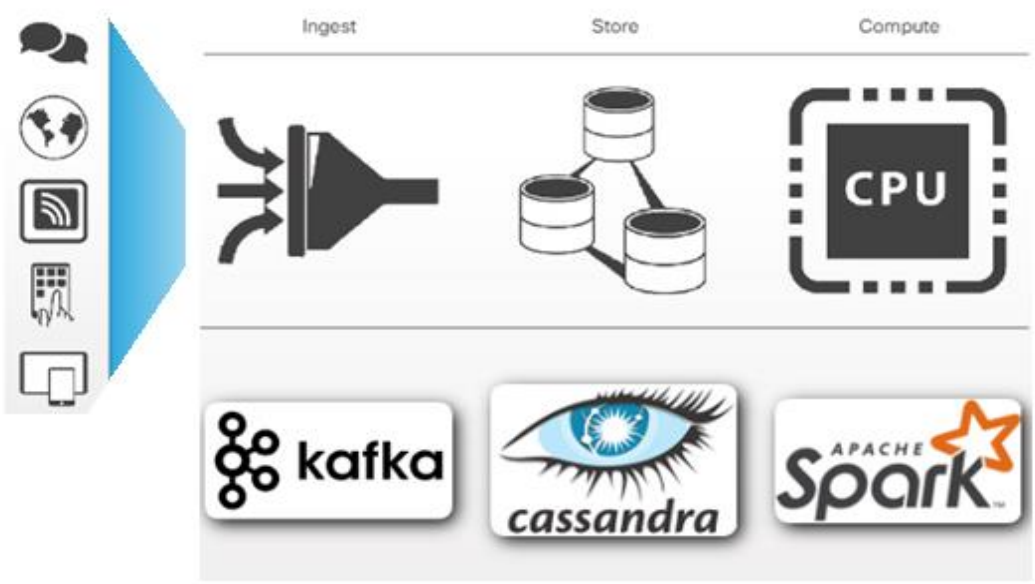


Рис. 16.1. Pipeline великих даних [1]

Великі дані часто надходять із багатьох різних джерел. Значна частина цих даних є потоковою і повинна прийматися в режимі реального часу. Масштабованість також є проблемою. Коли більше пристроїв підключається, більше даних потрібно приймати. Якість цих даних також може

бути проблемою. Дані можуть бути структурованими, неструктурованими або мати дуже складні формати. Існує багато різних інструментів для збору та переміщення великої кількості даних з різних джерел до сховища даних.

16.2. Розподілена потокова платформа Kafka

Для передачі даних у режимі реального часу необхідно використовувати розподілену потокову платформу, таку як Kafka. Характеристики Kafka:

- Потоки записів публікуються та підписуються на (pub-sub) аналогічно, як у системі обміну повідомленнями підприємства.
- Потоки записів стійкі до відмов через розподілене зберігання.
- Потоки записів обробляються, як вони трапляються, в режимі реального часу.

Kafka використовується для передачі потокової передачі даних у режимі реального часу між різними системами та програмами. Kafka також використовується для перетворення та реагування на потоки даних через додатки в режимі реального часу. Kafka була розроблена LinkedIn, але стала програмним забезпеченням з відкритим кодом у 2011 році. Cisco Systems, Netflix та eBay – це лише декілька підприємств, які використовують Kafka.

Розглянемо основні положення Kafka.

- Kafka працює на одному або декількох серверах як кластер. Сервери в кластері відомі як брокери.
- Кластер зберігає потоки записів у групах, що називаються темами.
- Кожен запис містить ключ, значення та часову позначку.

У Kafka є чотири основних API (рис. 16.2):

- Виробники. API, де потік записів публікується додатком на теми Kafka.
- Потокові процесори. API, де споживаються потоки входу з тем і можуть вироблятися потоки виводу до тем.
- З'єднувачі. API, де теми Kafka підключаються до існуючих систем та програмних додатків.
- Споживачі. API, де створюється потік записів у темах.

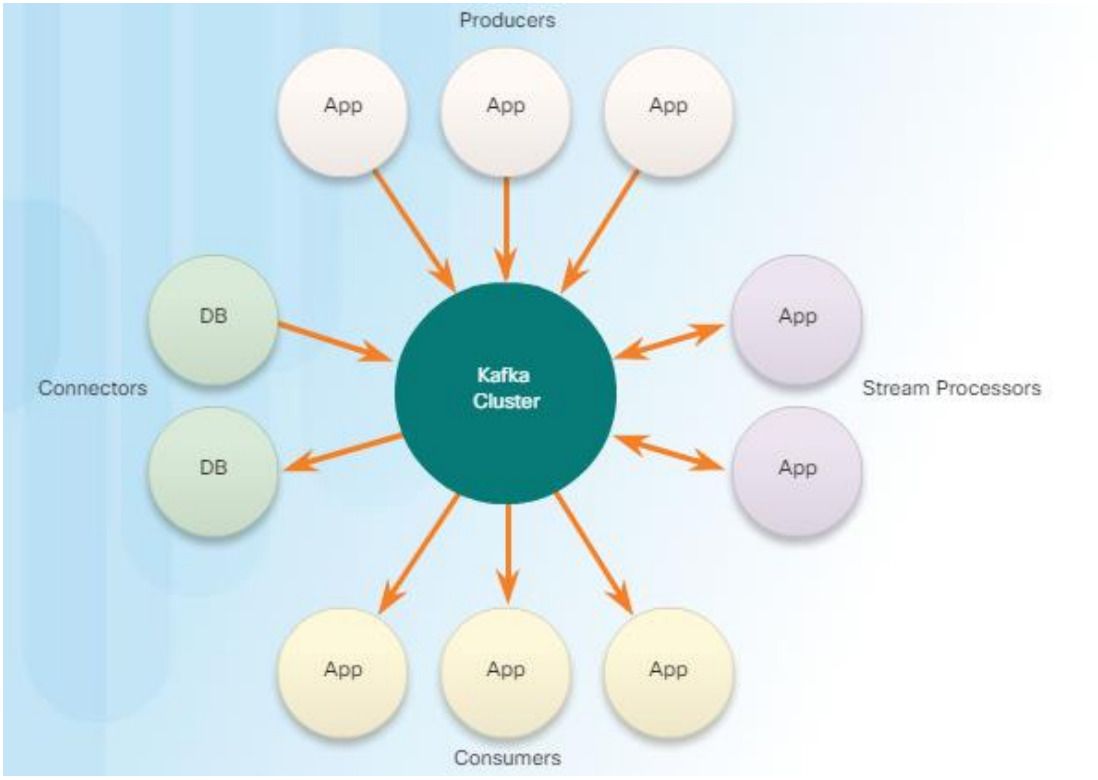


Рис. 16.2. Компоненти API Kafka [1]

Kafka діє як система обміну повідомленнями для централізації зв'язку між усіма виробниками та споживачами даних у системі. Kafka використовує розподілену обробку величезної кількості даних та масштабування, забезпечує відмовостійкість обладнання та програмного забезпечення. Передача повідомлень в Kafka відбувається двома способами (рис.16.3):

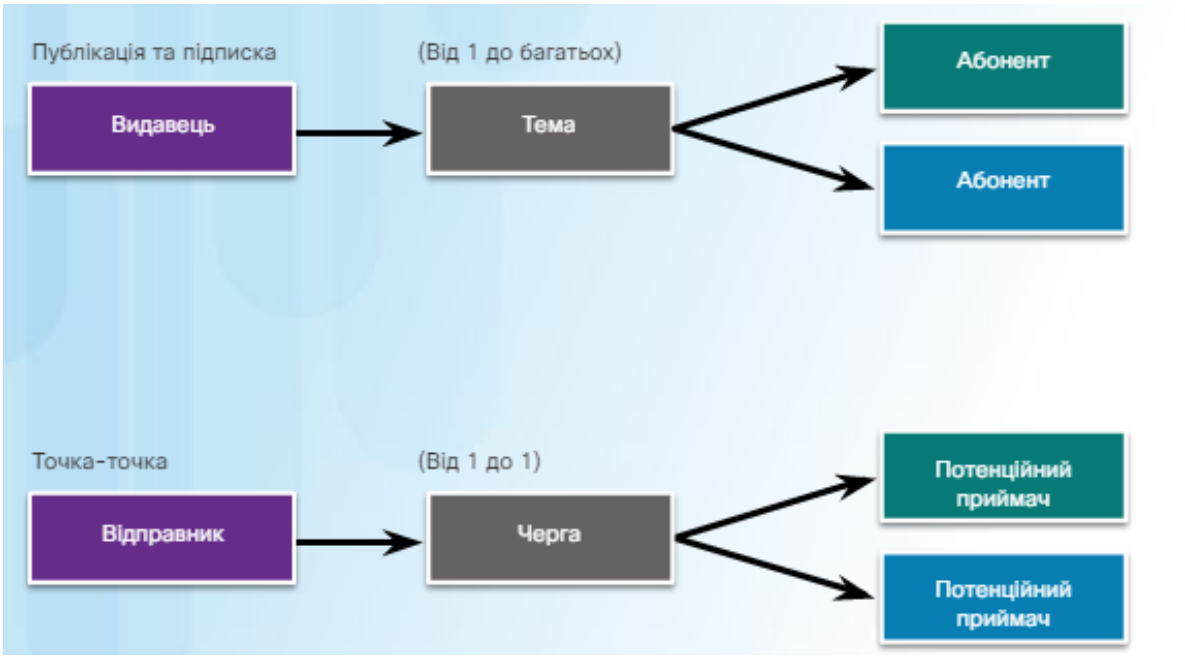


Рис. 16.3. Методи передачі повідомлень [1]

- **Опублікувати та підписатися** – запитувані повідомлення передаються всім споживачам.
- **Точка-в-точку** – кілька споживачів читають повідомлення з сервера. Кожне з цих повідомлень надходить до одного із споживачів.

Kafka – це набагато більше, ніж просто сервер обміну повідомленнями. Kafka працює аналогічно розподіленій базі даних.

Повідомлення, написані Kafka, копіюються на багато серверів і записуються на диск. Завдяки розподіленому проектуванню Kafka має високу доступність, підтримує автоматичне відновлення даних та відрізняється високою стійкістю до відмов мережі.

Kafka відрізняється від традиційних брокерів повідомлень використанням журналів транзакцій. Кожна тема складається з набору журналів, які називаються **розділами**. Виробники додають ці журнали, і споживачі можуть читати журнали, коли потрібно. **Теми** – це дані, які реплікуються багатьма брокерами для досягнення відмовостійкості.

Kafka може управляти високою пропускною здатністю завдяки тому, що багато задач покладаються на виробників та споживачів. Це зберігає брокери у легкій вазі та робить Kafka бажаним інструментом для багатьох платформ IoT. Завдяки величезній кількості даних, що надходять від датчиків та інших пристроїв у режимі реального часу, Kafka може регулювати прийом даних та надавати дані для кількох споживачів одночасно.

Згідно оцінок великих даних IBM, "кожен день ми створюємо 2,5 квинтільйонних байта даних" кожного дня, оскільки щохвилини:

- завантажується понад 300 годин відео YouTube;
- надсилається понад 3,5 мільйона текстових повідомлень;
- передається понад 86 тисяч годин відео Netflix;
- оцінюється понад 4 мільйони публікацій у Facebook.

Завдяки цьому постійно створюється величезний обсяг даних, навіть із хмарними сховищами, які доступні таким компаніям, як Amazon, Google, Microsoft, безпека даних стає великою проблемою.

Рішення Big Data повинні бути захищеними, мати високу стійкість до відмов, вміти горизонтально масштабувати та використовувати реплікацію, щоб дані не втрачалися.

Big Data – це термін для величезного обсягу даних, який ми постійно створюємо з величезної кількості джерел даних. Ці дані повинні бути релевантними, чистими та безпечними. Існує щонайменше п’ять проблем із зберіганням даних із Big Data:

- **Управління** – існує мало стандартів обміну даними та тисячі інструментів управління даними.
- **Безпека** – автентифікацію, доступ та облік важко забезпечити.
- **Неструктуровані дані** – неструктуровані дані важко аналізувати та шукати.

16.3. Переваги Cassandra

Cassandra – це система управління розподіленими базами даних з відкритим кодом NoSQL. База даних NoSQL не містить схем і не використовує традиційні методи зберігання та отримання даних, таких як реляційні бази даних, має просту конструкцію, підтримує горизонтальне масштабування та забезпечує більший контроль доступності. Cassandra може бути повністю розповсюджена та розгорнена по всьому світу, якщо потрібно та надає децентралізовану базу даних без жодної точки відмови.

Система управління розподіленими базами даних Cassandra розроблена компанією Facebook у 2008 році. Одна з речей, яка робить Cassandra швидкою, це те, що вона використовує послідовне читання та запис. Це дуже зручно для роботи з даними часових рядів в рішеннях IoT. Замість додавання або видалення даних у файлі створюється новий файл і видаляються старі файли.

Основні характеристики Cassandra:

- **Поширення даних** – дані можна копіювати в декількох центрах обробки даних.

- **Підтримка транзакцій** – підтримує атомність, консистенцію, ізоляцію та довговічність (atomicity, consistency, isolation, durability, ACID).
- **Еластична масштабованість** – коли потрібно більше даних або додається більше клієнтів, може бути додано більше обладнання для масштабування за потребою.
- **Швидке лінійне масштабування** – коли кількість вузлів збільшується в кластері, пропускна здатність збільшується, підтримуючи швидку реакцію.
- **Швидкий запис** – Cassandra виконує швидкий запис, зберігаючи сотні терабайт даних.
- **Завжди ввімкнено** – Cassandra завжди доступна навіть при збоях у апаратному та / або мережевому режимі.
- **Гнучкість зберігання даних** – підтримуються неструктуровані, структуровані та напівструктуровані дані.

Apache Hadoop визначає HDFS як "первинну систему зберігання, яка використовується програмами Hadoop", що дозволяє виконувати "надійні та швидкі обчислення".

Первинний NameNode в кластері регулює файлову систему та доступ до даних. У кластері також є DataNodes, часто одна фізична машина для обробки доданого сховища.

Дані зберігаються у файлах, поділяються на блоки та поширюються по DataNodes. HDFS копіює ці блоки на два додаткові сервери за замовчуванням.

Cassandra використовує файлову систему Cassandra File System (CFS).

Кожен вузол кластера має однакову однорангову реалізацію.

Кластери зберігають дані в реальному часі і аналітичні операції можуть бути виконані на цих даних (рис.16.4).

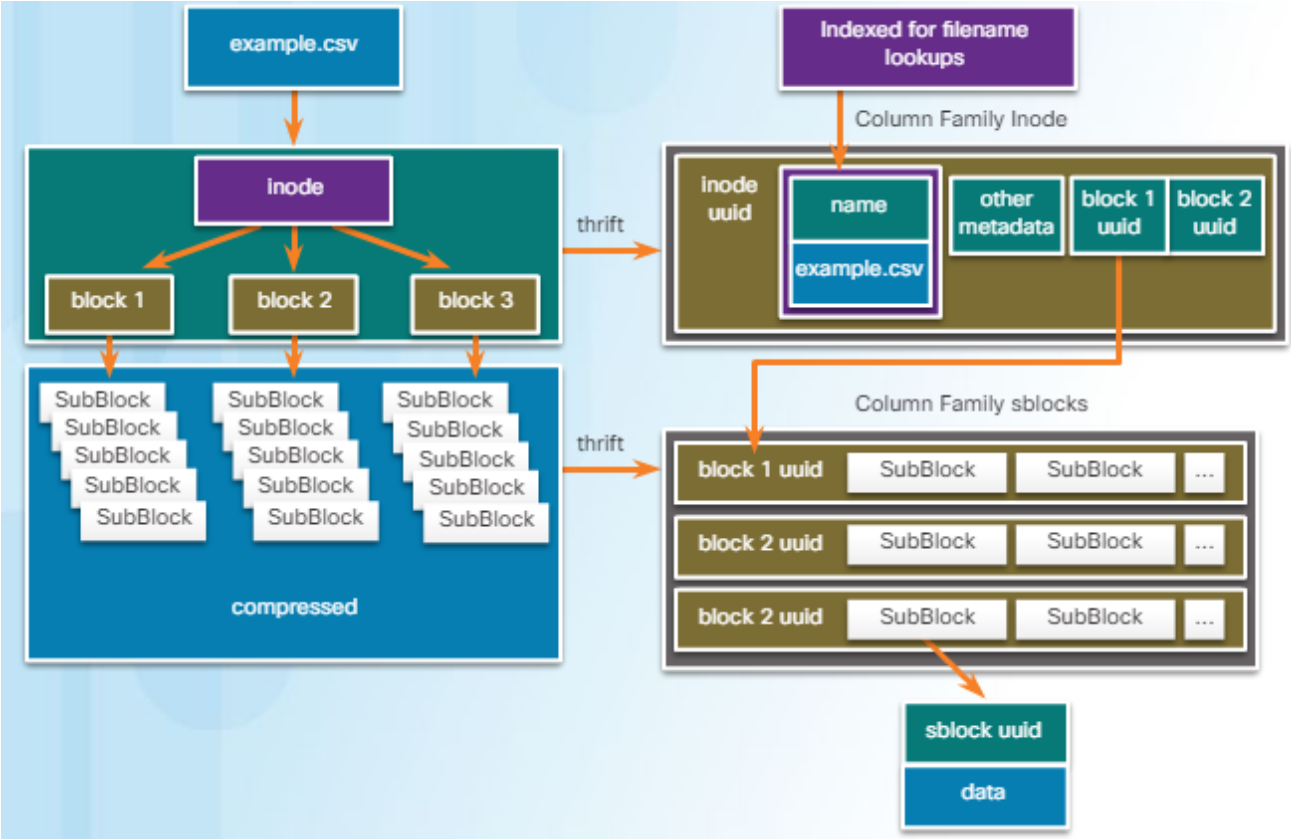


Рис. 16.4. Cassandra File System [1]

Вбудована реплікація копіює дані серед усіх реальних аналітичних та пошукових вузлів.

Є дві сімейства стовпців, схожі на таблиці RDBMS, які містять дані.

Дані в цих сімействах стовпців реплікуються через кластер для забезпечення захисту даних. Сімейства стовпців схожі з двома основними службами HDFS.

Сімейство стовпців **inode** займає місце служби HDFS NameNode.

Сімейство стовпців **sblocks** займає місце служби HDFS DataNode. Ці стовпці зберігають вміст будь-якого файлу.

Переваги використання CFS у порівнянні з HDFS:

- **Краща доступність** – рішення спільного зберігання не потрібно. Чим більше вузлів і кластерів, тим краща доступність. Це також можна покращити, збільшивши коефіцієнт реплікації, який визначає, скільки вузлів отримають репліковані дані.
- **Апаратна підтримка** – не потрібні спеціальні сервери та спеціальні мережеві пристрої для CFS.

- **Автоматичне відновлення** – вузли, кластери, навіть центри обробки даних можуть вийти з ладу, а дані про вузли та кластери залишатимуться доступними в інших місцях.
- **Інтеграція даних** – усі дані, записані в Cassandra, реплікуються як в аналітичні, так і в пошукові вузли. Це дозволяє одночасно виконувати аналітичні, пошукові роботи та завдання в режимі реального часу, не впливаючи один на одного.
- **Легше розгортання** – кластери легко налаштувати і можуть працювати лише за кілька хвилин.
- **Підтримується декілька центрів обробки даних** – CFS може запускати одну базу даних у кількох центрах обробки даних. Доступні інструменти, що дозволяють кожному центру обробки даних мати локальну копію всіх даних у базі даних. Аналітичні завдання можуть виконуватися в декількох центрах обробки даних одночасно.

HDFS – це відмінний вибір, коли для програми Hadoop потрібне недороге рішення для зберігання даних з акцентом на зберігання даних.

CFS може виконувати аналітику даних, що надходять із великих програм, що інтегруються в бази даних та СУБД.

Apache Cassandra повністю реалізує принципи доступності (Availability) і стійкості до поділу (Partition tolerance), забезпечуючи коректний відгук на будь-який запит і простоту масштабування. Однак узгодженість (Consistency), тобто несуперечливість даних вважається слабким місцем цієї розподіленої СУБД в зв'язку з її децентралізацією, коли на багатьох вузлах зберігаються різні репліки однієї і тієї ж інформації. Для усунення цієї проблеми використовується механізм налаштування рівнів узгодженості.

На відміну від NoSQL-бази даних для Big Data, Apache HBase, усі вузли кластера Cassandra рівноцінні – клієнти можуть з'єднуватися з будь-яким з них для запису і читання. Виконання запиту починається з його координації , щоб за допомогою ключа і міток визначити, на яких вузлах кластера знаходяться потрібні дані і відправити запит саме туди. Вузол, що виконує координацію,

називається **координатором (coordinator)** , а вузли, які обрані для збереження запису з даними ключем – **вузлами-реплік (replica nodes)**. Фізично координатором може бути один з вузлів-реплік.

Доступність даних безпосередньо залежить від рівня узгодженості операцій читання і запису, так як він визначає, скільки вузлів-реплік може відмовити при підтвердженні успішного виконання цих операцій. Наприклад, якщо рівень реплікації менший, ніж сума вузлів, з яких прийшло підтвердження про успішний записи даних, і з яких відбувається читання, тобто **гарантія суворої узгодженості (strong consistency)** . Сувора узгодженість гарантує, що після запису нового значення завжди буде прочитано. Інакше можлива ситуація, коли в результаті читання СУБД поверне застарілі дані. Для боротьби з цим в Apache Cassandra є **механізм підсумкової узгодженості (eventual consistency)** , який поширює дані по вузлах-реплік після того, як закінчиться координаційне очікування. Якщо при цьому будуть доступні не всі вузли-репліки, то доведеться задіяти інші засоби відновлення даних, зокрема, **читання з виправленням** і запускається вручну **анти-ентропійне відновлення вузла (anti-entropy node repair)**.

Рис. 16.5. Розподілений запис даних в кластері Apache Cassandra [2]

При запису даних рівень узгодженості визначає кількість вузлів-реплік, з яких буде очікуватися підтвердження вдалого закінчення операції – сигналу того, що дані успішно записані. Для запису існують такі рівні узгодженості:

- **ANY (0)** – дає можливість записати дані, навіть якщо всі вузли-репліки не відповідають. Координатор отримує відповідь від будь-якого одного вузла-реплік або дані збережуться за допомогою механізму спрямованої відправки (hinted handoff) на координатора.
- **ONE (1)** – координатор шле запити всіх вузлів-реплік, але повертає управління користувачеві, дочекавшись підтвердження від будь-якого першого вузла;
- **TWO (2)** – координатор чекає підтвердження від двох перших вузлів, перш ніж повернути управління;
- **THREE (3)** – координатор чекає підтвердження від трьох перших вузлів, перш ніж повернути управління;
- **QUORUM (4)** – координатор чекає підтвердження записів від більш ніж половини вузлів-реплік, а саме *round (N/2)+1*, де *N* – рівень реплікації;
- **ALL (5)** – координатор чекає підтвердження від всіх вузлів-реплік;
- **LOCAL_ QUORUM (6)** – координатор чекає підтвердження від більш ніж половини вузлів-реплік в тому ж центрі обробки даних, де розташований координатор. Це дозволяє позбутися затримок, пов'язаних з пересиланням даних у інші датацентри.
- **EACH_ QUORUM (7)** – координатор чекає підтвердження від більш ніж половини вузлів-реплік в кожному центрі обробки даних.

У разі читання рівень узгодженості визначає кількість вузлів-реплік, з яких буде зчитана інформація:

- **ONE (1)** – координатор шле запити до найближчого вузла-репліки, читаючи інші з метою виправлення (read repair) із заздалегідь заданою в конфігурації ймовірністю;
- **TWO (2)** – координатор шле запити до двох найближчих вузлів, вибираючи значення з більшою часовою позначкою;

- **THREE (3)** – координатор шле запити до трьох найближчих вузлів, вибираючи значення з більшою часовою позначкою;
- **QUORUM (4)** – збирається кворум, тобто координатор шле запити до більш ніж половині вузлів-реплік, а саме *round (N/2)+1* , де *N* – рівень реплікації;
- **ALL (5)** – координатор повертає дані після прочитання з усіх вузлів-реплік;
- **LOCAL_QUORUM** (6) – збирається кворум вузлів в тому датацентрі, де відбувається координація, і повертаються дані з останньою міткою часу;
- **EACH_QUORUM** (7) – координатор повертає дані після зборів кворуму в кожному з датацентрів.

Підводячи підсумок можливим рівням узгодженості в Apache Cassandra, можна зробити наступні висновки:

- **QUORUM** на читання і на запис забезпечить сувору узгодженість і баланс між затримкою на виконання цих операцій;
- strong consistency також буде досягнута під час запису ALL, і читанні ONE. Однак, в цьому випадку дані зчитуються швидше і з більшою доступністю. Для запису будуть потрібні всі робочі вузли-реплік.
- Зворотний випадок (**запис ONE, читання ALL**) забезпечує сувору узгодженість, запис буде виконуватися швидше і з більшою доступністю, тому що буде потрібно підтвердження про успішне закінчення операції лише з одного вузла.
- **При відсутності вимог про сувору узгодженість** можна прискорити операції читання і запису, а також покращити доступність за рахунок виставлення менших рівнів узгодженості.

Висновок до лекції 16

Розподілена потокова платформа Kafka відрізняється від традиційних посередників повідомлень використанням журналів транзакцій для передачі потокових даних у режимі реального часу між різними системами та програмами.

Cassandra – це система розподілених баз даних з відкритим кодом NoSQL. Cassandra використовує файлову систему Cassandra File System (CFS). Кожен вузол кластера має однакову однорангову реалізацію. Кластери зберігають дані в реальному часі і аналітичні операції виконуються на цих даних.

Питання для закріплення

1. Які проблеми прийому даних ви знаєте?
2. Для чого використовується Kafka?
3. Яке призначення та переваги Cassandra?

Список рекомендованої літератури

IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

Рівні узгодженості Apache Cassandra для розподіленої обробки Big Data // Електронний ресурс. Режим доступу: https://www.bigdataschool.ru/blog/cassandra-consistency-levels.html

What is Cassandra? // Електронний ресурс. Режим доступу: https://cassandra.apache.org/

About the Cassandra File System (CFS) – deprecated // Електронний ресурс. Режим доступу: https://docs.datastax.com/en/dse/5.1/dse-dev/datastax_enterprise/ analytics/ cfsAbout.html

Лекція 17.
Платформа Apache Spark

План лекції

17.1. Проблема обчислювальної функції.
17.2. Технологія Spark.
17.3. Порівняння Spark та MapReduce.
17.4. Spark і sparklyr для роботи з великими даними в R.

17.1. Проблема обчислювальної функції

Важливим завданням, з яким стикаються обчислення Big Data, є розмір наборів даних, що використовуються у різних галузях. Наприклад, генетичне секвенування (цифрове представлення геному людини) в останні роки стало

об’єктом дослідження наукової спільноти. Це пов’язано з тим, що високоефективні обчислення (high-performance computing, HPC) у поєднанні з розвитком технологій секвенування дали змогу за 26 годин секвенувати увесь геном людини. Послідовність усього геному людини становить близько 200 ГБ даних і вимагає величезної кількості обчислювальної потужності для роботи з нею. Кількість даних та їх аналіз у майбутньому може перевищити найпотужніший HPC. Коли це трапляється, HPC потрібно або масштабувати, додаючи на комп'ютер більше процесорів і пам'яті, або масштабувати, додаючи в кластер більше комп'ютерів та з'єднуючи їх швидкісними з'єднаннями. Коли обсяг обробки даних величезний, часті переміщення даних можуть значно збільшувати затримку. Бажано мати обчислювальну систему, яка працює для подолання обмежень системи зберігання, щоб звести затримку до мінімуму.

Аналітика, проведена на Big Data, може запропонувати нові можливості та непередбачувані тенденції, забезпечивши кращий огляд клієнтів та ринку загалом. Точна аналітика клієнтів, виявлення шахрайства та аналіз ризиків – це користь від аналізу великих даних. Ці складні обчислення потребують не тільки великих сховищ даних, але й низької затримки, високопропускної спроможності потокової обробки. Це ще більше збільшує розмір та складність аналітики HPC.

Щоб зменшити затримку від частих операцій вводу / виводу даних, обчислення можна перемістити на сервер, на якому розміщені дані. У багатьох місцях виконується кілька невеликих завдань для розподілу навантаження. Модель MapReduce може певною мірою зменшити велику кількість вузьких місць вводу-виводу та мереж, але це не головна мета MapReduce.

Затримка обчислень – це також проблема великих даних. Обчислення великих даних розділяється, щоб виконувати конкретні обчислення на різних вузлах. Коли один вузол працює повільніше, ніж інші, час відгуку збільшується. Це змушує результати загальної роботи чекати, коли цей вузол виконає свою частину, перш ніж їх можна буде реалізувати. Затримка є важливою проблемою у великих центрах обробки даних, що спричиняє значний вплив на обчислення, залежні від часу, наприклад, на потоках даних.

Також графік роботи може значно збільшити затримку. Часто великі обсяги роботи виконуються, тоді як виконуються інші, менші обсяги роботи.

Менші завдання, які повинні зачекати виконання великих завдань, можуть спричинити затримку. Це може бути проблемою, коли деякі завдання потрібно обробляти в режимі реального часу.

17.2. Технологія Spark

Spark – це відкритий, розповсюджений механізм обробки даних із відкритим кодом, який використовується для великих даних. Хоча платформа Hadoop дозволила багатьом компаніям успішно застосовувати парадигму MapReduce для розподілених обчислень величезних обсягів даних, кожен раз при виникненні нового завдання потрібно написання нового коду для операцій map і reduce, що є незручним і займає багато часу.

Для вирішення цієї проблеми в 2008 р. інженери з Facebook створили Hive – систему управління базами даних на основі Hadoop. Головною особливістю Hive стала підтримка SQL -подібних запитів до даних, що зберігаються в HDFS (цей новий діалект SQL отримав назву Hive Query Language, HQL).

У 2009 р. в Каліфорнійському університеті в Берклі був запущений дослідницький проект Spark з метою підвищити ефективність розподілених обчислень методом MapReduce і створити універсальну платформу для таких обчислень. У 2010 р. Spark був опублікований як проект з відкритим кодом, а в 2013 р. переданий фонду Apache Software Foundation. Spark використовує кешування в пам'яті для досягнення швидкої продуктивності, обмежуючи читання та запис дисків.

Apache Spark – фреймворк з відкритим вихідним кодом для реалізації розподіленої обробки неструктурованих і слабоструктурованих даних, що входить в екосистему проектів Hadoop. На відміну від класичного обробника з ядра Hadoop, що реалізує дворівневу концепцію MapReduce зі зберіганням проміжних даних на накопичувачах, Spark працює згідно парадигми резидентних обчислень (In-memory computing), обробляє дані в оперативній

пам'яті, завдяки чому дозволяє отримувати значний виграш в швидкості роботи для деяких класів задач, зокрема, можливість багаторазового доступу до завантажених в пам'ять даних робить бібліотеку привабливою для алгоритмів машинного навчання.

Spark може використовувати HDFS та досягати кращих показників, ніж MapReduce. Spark підтримує такі мови програмування, такі як Python, Scala, R та Java. Spark також підтримує структуровані бази даних на SQL. Завдяки цій різноманітності підтримуваних мов та систем можна реалізувати багато різних рішень Spark.

Spark включає бібліотеки, які допомагають створювати різні типи програм:

- **Shark SQL** – бібліотека SQL, яка ефективно підтримує складну аналітику, залишаючись стійкою до відмов.
- **Spark Streaming** – бібліотека обробки потоків, яка масштабується, підтримує високу пропускну здатність та підтримує відмовостійкість.
- **MLib** – масштабована, високоефективна бібліотека машинного навчання.
- **GraphX** – бібліотека, яка містить алгоритми теорії графів.

Однією з причин того, що Spark є настільки швидким, є те, що він використовує стійкі набори розподілених даних (Resilient Distributed Datasets, RDD) для зберігання вхідних, вихідних та проміжних даних у пам'яті замість диска. Це виключає вартість вводу-виводу. RDD є стійкими, тому що вони відстежують історію, якщо їм доведеться реконструювати себе після відмови. Вони розподіляються, тому що вони розповсюджені на багатьох вузлах кластера. Це дозволяє забезпечити надмірність, а також підвищити ефективність завдяки паралельній обробці. Коли запит надходить із програми, Spark створює та завантажує дані в RDD. Дані можуть надходити з будь-якого джерела, включаючи HDFS, CFS, AWS S3 або навіть базу даних SQL. Після створення RDD Spark може виконувати два різні типи операцій на RDD (рис.17.1):

- Трансформації – це будь-яке маніпулювання даними, включаючи відображення чи фільтрування. Будь-яка трансформація спричинить створення нового RDD. Оригінальний RDD не змінюється. Це дозволяє Spark здійснювати

контроль версій на RDD. Перетворення виконуються лише тоді, коли вони потрібні, наприклад, дією.

• Дії – дії взаємодіють із даними, але не створюють змін. Приклади включають агрегування, підрахунок або отримання конкретного елемента в RDD. Після запиту про дію створюється новий RDD, за яким слід виконати дію.

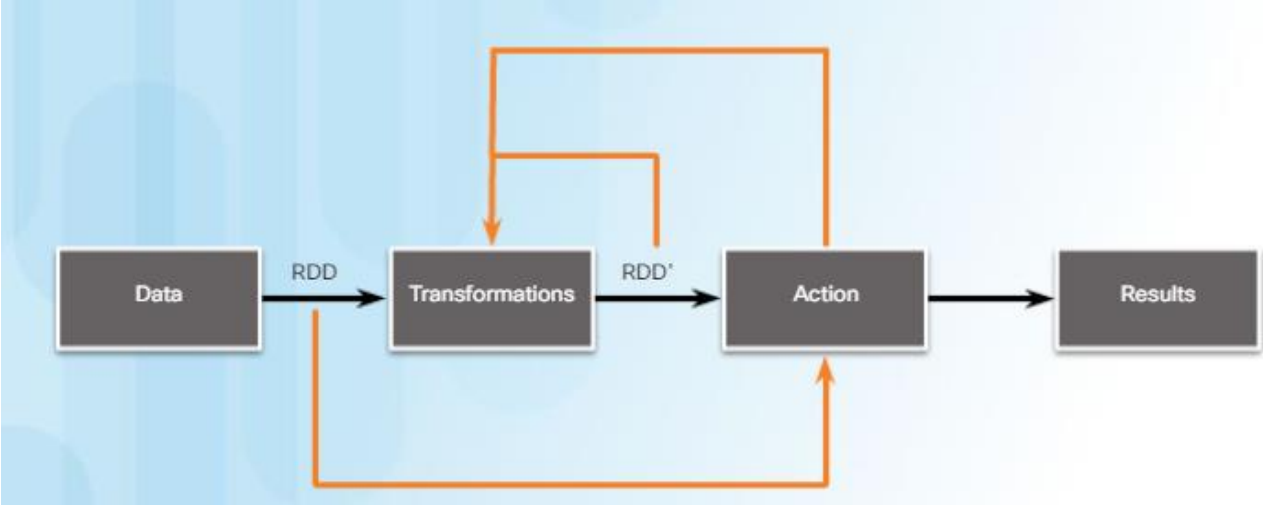


Рис. 17.1. Потік Spark [1]

17.3. Порівняння Spark та MapReduce

Spark може працювати безпосередньо над екземпляром Hadoop, використовуючи HDFS для зберігання та YARN для управління кластером.

Spark не потребує використання Hadoop. Можна використовувати інші рішення для зберігання, такі як CFS або AWS S3, а також інші менеджери кластерів, такі як Mesos.

Spark – це незалежна платформа, оскільки вона підтримує багато різних технологій та мов програмування. Це корисно, оскільки існує дуже багато різних потреб, коли йдеться про рішення та аналіз великих даних у багатьох різних сферах.

Підхід до рішення великих даних полягає у виборі правильних інструментів для роботи. Іноді поєднання Spark та MapReduce є найкращим рішенням для конкретної роботи. Використання Hadoop з MapReduce все ще є життєздатним при виконанні пакетної обробки за допомогою програми, яка використовує

виключно HDFS, або існуючі програми, які використовують цю технологію, вже у виробництві.

Spark набув великої популярності завдяки своїй продуктивності, простоті адміністрування, що при його використанні програмне забезпечення можна створювати швидше.

Розглянемо кілька причин використовувати Spark замість MapReduce при створенні програмних рішень для великих даних.

- Потокове передавання даних – Spark здатний обробляти величезні обсяги даних у режимі реального часу. Наприклад, дані можуть надходити з мобільних пристроїв, соціальних мереж або датчиків IoT.
- Неоднорідні дані – багато рішень для великих даних отримують дані з різних джерел.
- Машинне навчання – завдяки вбудованій бібліотеці машинного навчання Spark може задовольнити набагато більшу аудиторію, ніж попередні рішення.
- Додатки в режимі реального часу – обробка в пам'яті дозволяє Spark повертати результати обчислень набагато швидше, ніж MapReduce. Це важливо в усіх ситуаціях, але обов’язково, коли програма вимагає результатів у реальному часі.
- Менше коду – Spark підтримує багато різних мов програмування, а це означає, що менше коду потрібно писати та підтримувати.
- Досвід розробника – вивчати Spark набагато простіше, ніж MapReduce. Це набагато швидше приносить надійніший код для проекту.

17.4. Spark і sparklyr для роботи з великими даними в R

В екосистемі R є багато пакетів, що дозволяють користувачам працювати з віддаленими базами даних і виконувати масштабовані розподілені обчислення за допомогою призначених для цього програмних платформ, або фреймворків. Усі обчислення в системі R виконуються в оперативній пам'яті комп'ютера. Обсяг багатьох сучасних наборів даних (наприклад, зібраних

високонавантаженими веб-додатками і промисловими системами) набагато перевищує розмір оперативної пам'яті, доступний на окремому комп'ютері. Такі дані зазвичай зберігаються у віддаленій базі даних або в сховищі іншого типу. Залежно від завдання, для обробки подібних даних за допомогою R можна скористатися однією з наступних стратегій.

1. Створення репрезентативної вибірки обмеженого розміру. Практично усі класичні методи статистики припускають, що дослідник працює з репрезентативною вибіркою з деякою генеральною сукупністю. Тому для застосування таких методів цілком можна випадковим чином вибрати кілька тисяч або навіть сотень тисяч записів з бази даних, а потім локально виконати необхідний аналіз на комп'ютері користувача.

Такий підхід особливо корисний на початкових стадіях проекту, коли потрібно швидко ознайомитися з властивостями даних або створити прототип моделі. При цьому користувачеві буде доступно все розмаїття існуючих для R додаткових пакетів. Недолік цього підходу полягає в тому, що іноді створити репрезентативну вибірку буває складніше, ніж здається. Працюючи ж з вибіркою, дослідник ризикує зробити невірні висновки щодо властивостей генеральної сукупності.

2. Розбиття даних на частини. Рішення ряду завдань передбачає обробку логічно розділених наборів даних, наприклад, відповідних певним часовим періодам, окремим географічним областям, компаніям, користувачам тощо. Такі набори можна легко витягти з бази даних і далі послідовно (іноді паралельно) обробити за допомогою R на локальному комп'ютері дослідника. У цій стратегії аналізу піддаються усі наявні дані. Однак ці дані повинні бути нероздільні на окремі частини, що не завжди можливо або не завжди має сенс. Більш того, в залежності від обсягу даних, такий підхід може зайняти занадто багато часу і обчислювальних ресурсів.

3. Виконання ресурсномістких обчислень на стороні бази даних. Часто перед виконанням статистичного аналізу або побудовою прогнозної моделі до даних необхідно застосувати такі операції, як фільтрація, групування, агрегування

тощо. Більшість баз даних чудово справляються з такими операціями і тому має сенс спочатку зробити подібні обчислення саме на стороні бази даних, а потім вже завантажити отриманий набір даних меншого розміру на комп'ютер користувача і провести його подальшу обробку за допомогою R.

До цієї стратегії варто віднести також можливість побудови деяких прогнозних моделей з використанням обчислювальних ресурсів бази даних. Наприклад, пакет **modeldb** дозволяє таким чином створювати моделі лінійної регресії. Тут же варто згадати і спроби деяких компаній реалізувати можливість виконання R-скриптів повністю на стороні бази даних. Зокрема, така можливість є в Microsoft SQL Server. Залежно від використовуваної бази даних, необхідна користувачу функціональність іноді може бути недоступна. Більш того, не всі бази даних однаково ефективні: виконання ресурсоємних запитів може привести до істотного уповільнення деяких з них, що, в свою чергу, може викликати і інші небажані наслідки (наприклад, уповільнення роботи веб-сайту, який обслуговується подібною базою даних).

4. Використання спеціалізованих програмних платформ для роботи з великими даними. Якщо описані вище стратегії з тих чи інших причин не підходять, варто звернутися до спеціалізованих програмних платформ, призначених для організації і виконання розподілених обчислень над великими даними. Найбільш відомими і широко використовуваними серед таких платформ є Hadoop і Spark (обидві входять до складу проектів фонду Apache Software Foundation і тому їх також часто називають Apache Hadoop і Apache Spark).

В R є кілька пакетів (зокрема, **sparklyr**), що надають зручний інтерфейс для роботи з цими платформами. Spark є однією з найбільш використовуваних платформ для роботи з великими даними, яка характеризується швидкодією за рахунок виконання обчислень в оперативній пам'яті великої кількості комп'ютерів, об'єднаних в один кластер, а також завдяки ефективним протоколам передачі даних по мережі. Пакет **sparklyr** надає зручний інтерфейс для роботи зі Spark-кластерами з середовища R. Зокрема, з його допомогою можна встановлювати з'єднання з кластером; виконувати операції перетворення,

фільтрації і агрегування даних з використанням синтаксису **dplyr**; будувати прогнозні моделі з використанням алгоритмів машинного навчання, реалізованих в бібліотеці MLlib для Spark; працювати з іншими R-пакетами, які використовують Spark для виконання розподілених обчислень.

Завдяки таким пакетам, як **sparklyr**, ми можемо використовувати R в якості клієнта, який відправляє обчислювальні завдання (Push compute) на Spark-кластер, а потім збирає результати обчислень (Collect results) для подальшого аналізу вже в самій системі R. Є два способи формального уявлення подібних обчислювальних задач перед їх відправкою на кластер: або з використанням SQL-команд, або за допомогою функцій з пакета **dplyr**. Хоча SQL (точніше, Spark SQL, який, в свою чергу, заснований на діалекті HiveQL) дозволяє формулювати як завгодно складні операції над даними, в більшості стандартних випадків **dplyr** більш зручний, оскільки він добре знайомий більшості сучасних користувачів R і володіє більш лаконічним синтаксисом.

У наведених нижче прикладах використані дані з пакета nycflights13, який містить кілька таблиць з описом 336776 авіарейсів з аеропортів Нью-Йорка в 2013 р. Запустимо локальний Spark-кластер і завантажимо в нього необхідні таблиці [4]:

```
require(sparklyr)
require(nycflights13)
# підключення до кластеру:
sc <- spark_connect(master = "local", version = "2.3")
# завантаження даних в кластер:
flights_tbl <- copy_to(sc, flights, "flights") # дані по рейсам
airlines_tbl <- copy_to(sc, airlines, "airlines") # дані по авіакомпаніям
```

Припустимо, що перед нами стоїть завдання побудувати модель, яка передбачає ймовірність прибуття затриманого рейсу без запізнення (за умови затримки вильоту на 15-30 хвилин). Будемо розглядати це завдання як випадок бінарної класифікації: цікавий для нас відгук приймає значення 1 якщо затриманий рейс прибув без запізнення, і 0 якщо немає.

Процес побудови прогнозних моделей включає кілька кроків, першим з яких є підготовка та розвідувальний аналіз даних. Сам розвідувальний аналіз також може складатися з декількох кроків, таких як виявлення і усунення проблем з якістю аналізованих даних (пропущені спостереження, викиди тощо), розрахунок описових статистик, виявлення найбільш перспективних предикторів для подальшого включення в модель і тощо.

Подивимося, як з цим можуть допомогти команди з пакета **dplyr**.

За допомогою пакета **dplyr** можна виконувати такі стандартні типи обчислень на Spark-кластері:

- вибір, фільтрація і агрегування змінних;
- використання віконних функцій;
- об'єднання декількох таблиць за допомогою join-операторів;
- імпорт результатів обчислень з Spark в середовище R.

До найбільш важливих команд **dplyr** відносяться select(), filter(), mutate(), summarise() і arrange(). Крім того, для виконання групових операцій використовується команда group_by().

Ці та інші команди dplyr можна об'єднувати в "ланцюжки" за допомогою оператора %>% з пакета **magrittr** (підвантажується одночасно з dplyr і тому окремо його викликати не потрібно). Підрахуємо загальну кількість польотів, виконаних кожною авіакомпанією за 2013 р., а потім вибираємо 5 авіакомпаній з найбільшим числом польотів:

```
require(dplyr)
flights_tbl %>%
group_by(carrier) %>%
summarise(N = n()) %>%
arrange(desc(N)) %>%
head(5)
## # Source:     spark<?> [?? x 2]
## # Ordered by: desc(N)
```

```
##   carrier     N
##   <chr>   <dbl>
## 1 UA      58665
## 2 B6      54635
## 3 EV      54173
## 4 DL      48110
## 5 AA      32729
```

Оскільки Spark "розуміє" тільки SQL, то в дійсності усі обчислення, задані в R за допомогою команд **dplyr** перед відправкою на Spark-кластер автоматично переводяться на SQL наступним чином:

select () -> SELECT

filter () -> WHERE

mutate () -> оператори + , - , / , * , log та ін.

summarise () -> функції агрегування SUM , MIN , MAX і ін.

arrange () -> ORDER

group_by () -> GROUP BY

Крім того, dplyr автоматично переводить на SQL такі базові команди R:

Математичні оператори:

+, -, *, /, %%, ^

Математичні функції:

abs, acos, asin, asinh, atan, atan2, ceiling, cos, cosh, exp,

floor, log, log10, round, sign, sin, sinh, sqrt, tan, tanh

Логічні оператори:

<, <=, !=, >=, >, ==, %in%

Булеві оператори:

&, &&, |, ||, !

Функції для роботи с символьними рядками:

paste, tolower, toupper, nchar

Функції для перетворення типу змінної:

as.double, as.integer, as.logical, as.character, as.date

```
# Функції агрегування:
mean, sum, min, max, sd, var, cor, cov, n
```

Функція show_query () з пакету **dplyr** дозволяє переглянути SQL-запит, який формується з відповідного коду R. Для наведеного вище прикладу отримуємо:

```
flights_tbl %>%
   group_by(carrier) %>%
   summarise(N = n()) %>%
   arrange(desc(N)) %>%
   head(5) %>%
   show_query()
## <SQL>
## SELECT `carrier`, count(*) AS `N`
## FROM `flights`
## GROUP BY `carrier`
## ORDER BY `N` DESC
## LIMIT 5
```

Як і у випадку з базами даних, при роботі з Spark **dplyr** дотримується принципу "ледачих обчислень" (Lazy evaluation). Це означає, що всі обчислення на Spark-кластері відкладаються до останнього моменту, поки не знадобиться їх результат. Крім того, підсумковий результат обчислень буде імпортовано в середовище R тільки якщо користувач запросить його в явному вигляді. Наприклад, наведену вище послідовність команд ми могли б прописати в такий спосіб:

```
command_1 <- group_by(flights_tbl, carrier)
command_2 <- summarise(command_1, N = n())
command_3 <- arrange(command_2, desc(N))
command_4 <- head(command_3, n = 5)
```

Жодна з цих послідовних команд не буде виконана до тих пір, поки ми не запитаємо результат обчислень. Як зазначалося раніше , в об'єктах command_1,

command_2 ... command_4 зберігається тільки інформація, необхідна для підключення до Spark-кластеру, а також інструкції для відповідних обчислень.

Під "запитом результату обчислень в явному вигляді" розуміються дві ситуації: або висновок результату на екран, або збереження його в локальний об'єкт R:

```
# вивід на екран:
command_4
## # Source:     spark<?> [?? x 2]
## # Ordered by: desc(N)
##   carrier     N
##   <chr>   <dbl>
## 1 UA      58665
## 2 B6      54635
## 3 EV      54173
## 4 DL      48110
## 5 AA      32729
# імпорт результату та збереження в об'єкт R:
result <- collect(command_4)
result
## # A tibble: 5 x 2
##   carrier     N
##   <chr>   <dbl>
## 1 UA      58665
## 2 B6      54635
## 3 EV      54173
## 4 DL      48110
## 5 AA      32729
```

У другому випадку ми застосували спеціальну команду collect () з пакету dply. У певному сенсі collect () протилежна copy_to (), яка була використана вище для імпорту даних з R в Spark.

У пакеті **dplyr** є кілька функцій для виконання стандартних JOIN -операцій над двома таблицями: inner_join () , left_join () , right_join () , full_join () , semi_join (), nested_join () і anti_join (). У наступному прикладі виконаний LEFT JOIN таблиці flights_tbl з таблицею airlines_tbl за полем carrier:

```
flights_tbl %>%
left_join(airlines_tbl, by = "carrier") %>%
glimpse()
## Observations: ??
## Variables: 20
## Database: spark_connection
## $ year          <int> 2013, 2013, 2013, 2013, 2013, 201...
## $ month         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day           <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time      <int> 517, 533, 542, 544, 554, 554, 555...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600...
## $ dep_delay     <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, ...
## $ arr_time      <int> 830, 850, 923, 1004, 812, 740, 91...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 85...
## $ arr_delay     <dbl> 11, 20, 33, -18, -25, 12, 19, -14...
## $ carrier       <chr> "UA", "UA", "AA", "B6", "DL", "UA...
## $ flight        <int> 1545, 1714, 1141, 725, 461, 1696,...
## $ tailnum       <chr> "N14228", "N24211", "N619AA", "N8...
## $ origin        <chr> "EWR", "LGA", "JFK", "JFK", "LGA"...
## $ dest          <chr> "IAH", "IAH", "MIA", "BQN", "ATL"...
## $ air_time      <dbl> 227, 227, 160, 183, 116, 150, 158...
## $ distance      <dbl> 1400, 1416, 1089, 1576, 762, 719,...
## $ hour          <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, ...
## $ minute        <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0...
## $ time_hour     <dttm> 2013-01-01 10:00:00, 2013-01-01 ...
## $ name          <chr> "United Air Lines Inc.", "United ...
```

Хоча стандартні команди dplyr і деякі базові функції R автоматично перетворюються на SQL, їх буде недостатньо для розробки більш складних обчислень над даними за допомогою Spark.

Spark SQL заснований на мові запитів Hive Query Language (HiveQL) і всі функції цієї мови можна використовувати в поєднанні з командами **dplyr**.

Наприклад, для обчислення медіанного значення змінної dep_delay (затримка рейсу, хв.) з таблиці flights_tbl ми не можемо скористатися базовими функціями R median () або quantile (), це призведе до помилки. Але ми можемо застосувати Hive-функцію percentile ():

```
flights_tbl %>%
summarise(median = percentile(dep_delay, 0.5))
## # Source: spark<?> [?? x 1]
##   median
##    <dbl>
## 1     -2
```

Коли при перекладі коду R на SQL dplyr зустрічає незнайому функцію, то він просто включає її в SQL-запит "як є":

```
flights_tbl %>%
summarise(median = percentile(dep_delay, 0.5)) %>%
show_query()
## <SQL>
## SELECT percentile(`dep_delay`, 0.5) AS `median`
## FROM `flights`
```

Hive-функція percentile() дозволяє одночасно обчислити кілька процентилей. Для цього на неї потрібно подати масив (array ()) з необхідними значеннями процентилей:

```
flights_tbl %>%
  summarise(perc = percentile(dep_delay, array(0.25, 0.5, 0.75)))
# Source: spark<?> [?? x 1]
  perc
  <list>
1 <list [3]>
```

Результатом виконання наведеної команди є список з трьома значеннями. Щоб автоматично отримати ці значення зі списку, використовується Hive-функція explode () :

```
flights_tbl %>%
  summarise(perc = percentile(dep_delay, array(0.25, 0.5, 0.75))) %>%
  mutate(perc = explode(perc))
## # Source: spark<?> [?? x 1]
##    perc
##   <dbl>
## 1    -5
## 2    -2
## 3    11
```

Застосуємо отримані знання в ході розвідувального аналізу даних з таблиці flights_tbl. З'ясуємо, чи є в наших даних пропущені значення. Це можна зробити декількома способами. Нижче підрахунок пропущених значень виконаний для всіх стовпців таблиці flights_tbl за допомогою команди summarise_each () з пакету **dplyr** в поєднанні з анонімною функцією, яка задає логіку обчислень:

```
  flights_tbl %>%
  summarise_each(list(~sum(as.integer(is.na(.))))) %>%
  glimpse
## Observations: ??
## Variables: 19
## Database: spark_connection
## $ year          <dbl> 0
## $ month         <dbl> 0
## $ day           <dbl> 0
## $ dep_time      <dbl> 8255
## $ sched_dep_time <dbl> 0
## $ dep_delay     <dbl> 8255
## $ arr_time      <dbl> 8713
## $ sched_arr_time <dbl> 0
## $ arr_delay     <dbl> 9430
## $ carrier       <dbl> 0
## $ flight        <dbl> 0
## $ tailnum       <dbl> 2512
## $ origin        <dbl> 0
```

```
## $ dest          <dbl> 0
## $ air_time      <dbl> 9430
## $ distance      <dbl> 0
## $ hour          <dbl> 0
## $ minute        <dbl> 0
## $ time_hour     <dbl> 124
```

Як бачимо, пропущені значення дійсно мають місце в деяких змінних. Максимальна кількість пропущених значень досягає 9430, що становить менше 3% від загального числа спостережень в таблиці (336776). Розмірність таблиці можна з'ясувати за допомогою команди sdf_dim () з пакету **sparklyr**, яка є аналогом базової R-функції dim ().

У пакеті **sparklyr** є ряд інших команд, ім'я яких починається на sdf_ (від "Spark data frame"), наприклад, sdf_nrow () , sdf_ncol () , sdf_bind_rows () , sdf_pivot (). Так як частка пропущених значень невелика, ми можемо видалити відповідні рядки з таблиці без особливого ризику вплинути на якість подальшого аналізу. Для цього скористаємоя базовою функцією na.omit ():

```
flights_full <- flights_tbl %>% na.omit()
## * Dropped 9554 rows with 'na.omit' (336776 => 327222)
flights_full %>% sdf_dim()
## [1] 327222     19
```

Оскільки нас цікавлять рейси, затримка яких склала від 15 до 30 хв. (включно), потрібно відфільтрувати дані відповідним чином.

Паралельно додамо новий стовпець target зі значеннями залежної змінної:

```
flights <- flights_full %>%
  filter(dep_delay >= 15, dep_delay <= 30) %>%
  mutate(target = as.integer(arr_delay <= 0))
# розмірність таблиці:
flights %>% sdf_dim()
## [1] 24507    20
```

Ми отримали таблицю з 24507 рядками і 20 стовпцями. Це невелика таблиця і вже зараз її можна було б імпортувати з Spark в R (за допомогою команди collect ()) для подальшого аналізу. Однак з метою демонстрації можливостей роботи Spark з R ми залишимо цю таблицю в пам'яті кластера.

З'ясуємо, які з наявних змінних корелюють з залежною змінною target. Логічно було б очікувати, що ймовірність прибуття затриманого рейсу за розкладом в значній мірі визначається відстанню між аеропортом вильоту і аеропортом прибуття (стовпець distance, що виражається в милях).

Розрахуємо медіанне значення цієї відстані для обох класів залежною змінною:

```
flights %>%
group_by(target) %>%
summarise(median_dist = percentile(distance, 0.5))
## # Source: spark<?> [?? x 2]
##  target median_dist
##   <int>       <dbl>
## 1      0         820
## 2      1        1089
```

Дійсно, чим більше відстань між аеропортами, тим більше шанс у затриманого рейсу надолужити згаяний час і прибути без запізнення.

Висновок до лекції 17

Apache Spark – це уніфікований механізм аналітики з відкритим кодом для широкомасштабної обробки даних.

Spark забезпечує інтерфейс для програмування цілих кластерів з неявною паралельністю даних та стійкістю до відмов.

Spark підтримує різні мови програмування, такі як Python, Scala, R та Java.

Apache Spark також підтримує структуровані бази даних.

Питання для закріплення

1. У чому полягає проблема обчислювальної функції?
2. Опишіть особливості технології Spark.
3. Яка відмінність між Spark та MapReduce?
4. Які пакети в R використовуються для роботи з Spark?

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

2. Apache Spark // Електронний ресурс. Режим доступу: https://spark.apache.org/

3. Spark і sparklyr для роботи з великими даними в R // Електронний ресурс. Режим доступу: https://r-analytics.blogspot.com/2020/02/spark-intro.html

4. tidyverse/nycflights13 // Електронний ресурс. Режим доступу: https://github.com/tidyverse/nycflights13

5. Локальний Spark-кластер // Електронний ресурс. Режим доступу: https://r-analytics.blogspot.com/2020/02/spark-r-connect.html

6. Аналіз даних в Spark-кластері за допомогою пакета dplyr // Електронний ресурс. Режим доступу: https://r-analytics.blogspot.com/2020/03/spark-dplyr.html

Лекція 18.

Lambda та Kappa архітектури оброблення великих даних

План лекції

18.1. Lambda - архітектура.
18.2. Переваги і недоліки Lambda -архітектури.
18.3. Kappa - архітектура.
18.4. Переваги і недоліки Kappa-архітектури.

18.1. Lambda - архітектура

Прагнучи зблизити аналіз «історичних» даних та даних, отримуваних у режимі реального часу [1], була створена архітектура Lambda (рис. 18.1).

Lambda – це архітектура обробки даних, яка використовує як обробку потоків, так і пакетну обробку для отримання точного перегляду як "живих" даних, так і пакетних даних.

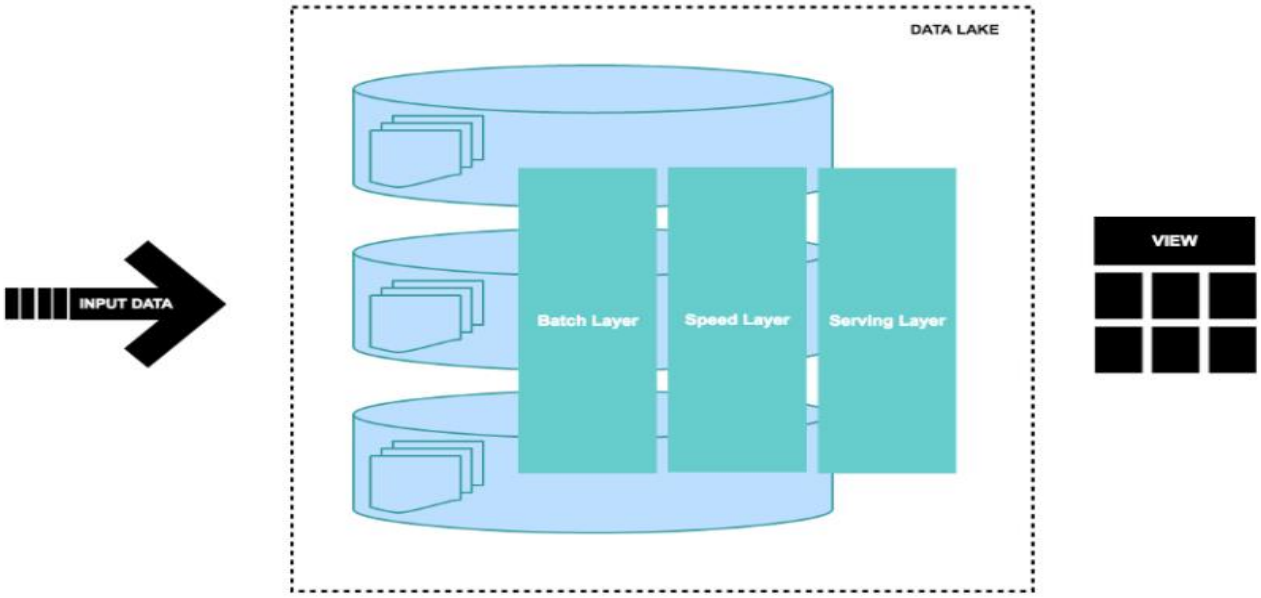


Рис. 18.1. Узагальнена Lambda – архітектура [2]

Lambda -архітектура має чотири шари (рис. 18.2):

Ingestion – цей рівень імпортує дані. Це можуть бути дані з багатьох джерел, включаючи потоки даних.

Batch – це дані в рівні спокою. Дані тут часто будуються за графіком і включають імпорт даних із потокового шару. Точність важливіша за швидкість.

Stream – це складний шар, що підтримує поступове оновлення. Низька затримка тут має більший пріоритет, ніж точність.

Presentation – цей шар запускає операцію та приймає всі запити і використовує швидкісний або пакетний шар.

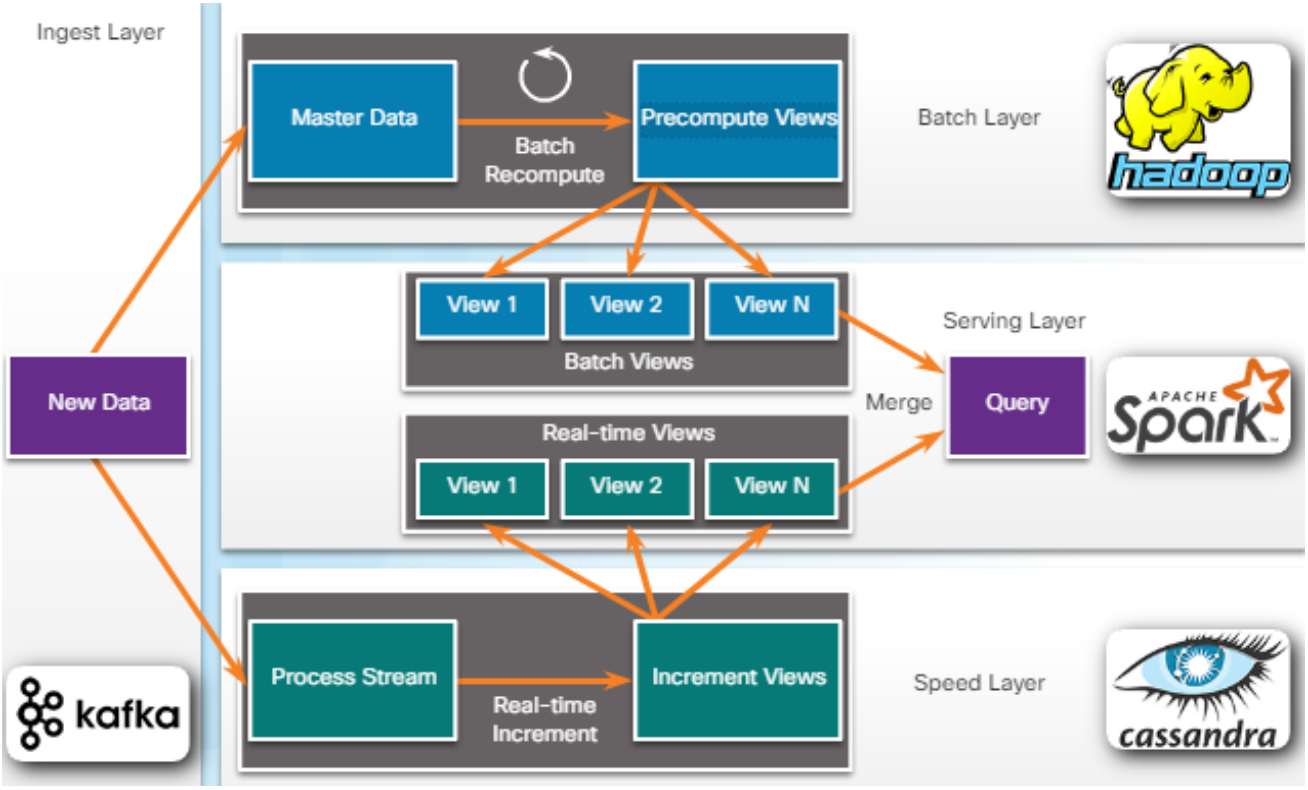


Рис. 18.2. Технології Lambda – архітектури [1]

Усі дані, що надходять, надсилаються одночасно і в пакетний шар, і в швидкісний шар. Пакетний шар управляє набором основних даних і попередньо обчислює пакетні представлення, які постійно обчислюються. Обслуговуючий шар індексує перегляди пакетів, щоб їх можна було запитати.

Швидкість шару стосується лише останніх даних, що компенсує затримку оновлень рівня, який обслуговує дані. Два різні результати запиту можуть бути об'єднані, щоб сформувати новий тип даних.

IoT є ідеальною областю для реалізації архітектури Lambda. Дані генеруються з великою швидкістю, набори даних можуть бути дуже великими, а запити можуть бути як для даних у спокої, так і для даних у русі.

Розглянемо приклад із реального життя використання архітектури Lambda для візуалізації 171 автобусних маршрутів у Лос-Анджелесі, Каліфорнія [1]. Lambda включає SACK (Spark, Akka, Cassandra, Kafka). Це дозволяє здійснювати аналітику даних у режимі реального часу.

Akka – це безкоштовний інструмент з відкритим кодом для створення розподілених та стійких програм, керованих повідомленнями [3].

Akka використовується для отримання метаданих інформації про маршрут та зберігання їх у Cassandra кожні 30 секунд. У цьому прикладі Akka для запиту даних використовує безкоштовний API REST. Далі дані зберігаються в Cassandra. Подальші запити надсилаються до Kafka. Потім використовується Spark для зчитування інформації про транспортний засіб від Kafka. Коли всі ці дані доступні, інший API використовується для візуалізації положення шин за допомогою OpenStreetMap. Поточні положення автобусів передаються прямо з Kafka на карту за допомогою веб-розмови [3, 4].

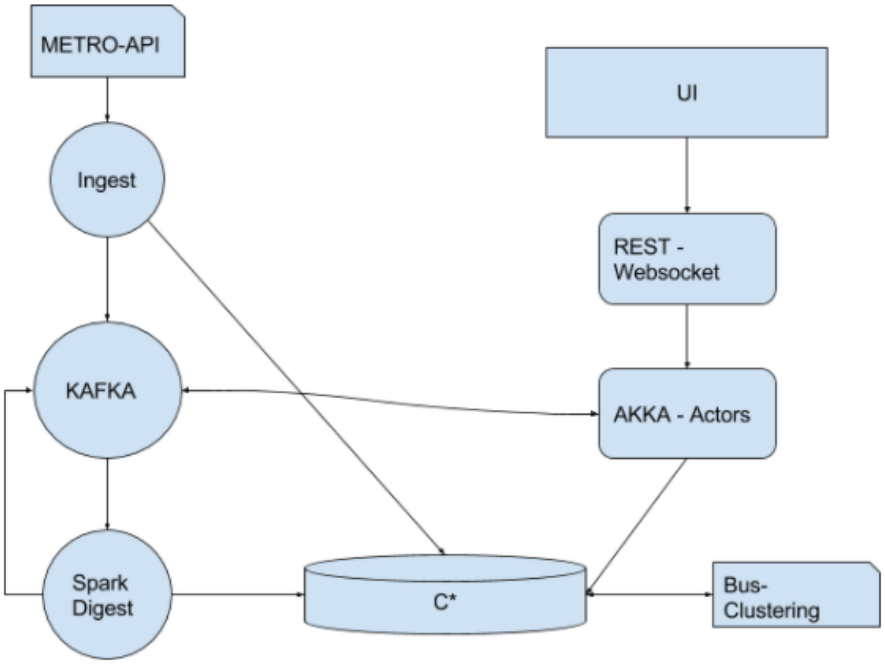


Рис. 18.3. Архітектура IoT Analytics Platform (Ingest – Akka, Digest – Spark, UI – Javascript, Openstreetmap, Backend – Akka) [3]

Ця платформа не обмежується лише даними IoT. Вона підходить для рішень, де є багато вхідних даних. Це дуже корисно, коли є кілька паралельно вхідних потоків даних. Це краще рішення, ніж традиційні RDBMS, які не зможуть надати дані в реальному часі.

Lambda -архітектура може бути розгорнута для тих корпоративних моделей обробки даних, де призначені для користувача запити повинні оброблятися з використанням незмінного сховища даних; потрібні швидкі відповіді, система повинна бути здатна обробляти різні оновлення у формі нових потоків даних;

жоден зі збережених записів не повинен бути видалений, і це повинно дозволити додавати оновлення та нові дані в базу даних. Такі компанії, як Twitter, Netflix і Yahoo, використовують цю архітектуру для відповідності стандартам якості обслуговування.

18.2. Переваги і недоліки Lambda –архітектури

Lambda -архітектура має наступні переваги для Big Data системи:

- збереження історичних даних за рахунок пакетного рівня на базі Hadoop Data Lake або іншого відмовостійкого розподіленого сховища з низькою ймовірністю помилок і збоїв;
- баланс швидкості і надійності;
- масштабованість.

Однак, Lambda -підходу властиві такі недоліки:

- неможливість зміни стратегії аналізу даних «на льоту» – кінцева несуперечливість даних унеможливлює відправку інформації назад на пакетний рівень. Потрібне повторне проведення всіх обчислень, через прив'язку до сховища, дані важко перенести або реорганізувати;
- більшість інструментів, орієнтованих на Lambda-архітектуру, відносяться до NoSQL та не підтримують SQL-запити або інші засоби бізнес-аналітики;
- складність – багато різнорідних компонентів, які передають дані один одному, що затримує обчислення в реальному часі, логіка обробки інформації дублюється з використанням різних структур даних, ускладнюючи загальне управління. Також збільшуються накладні витрати на розробку.

Такі недоліки частково компенсуються за допомогою Apache Spark.

Однак для швидкої обробки подій в режимі реального часу без пакетного представлення більш доцільний інший підхід – Kappa – архітектура (рис.18.4).

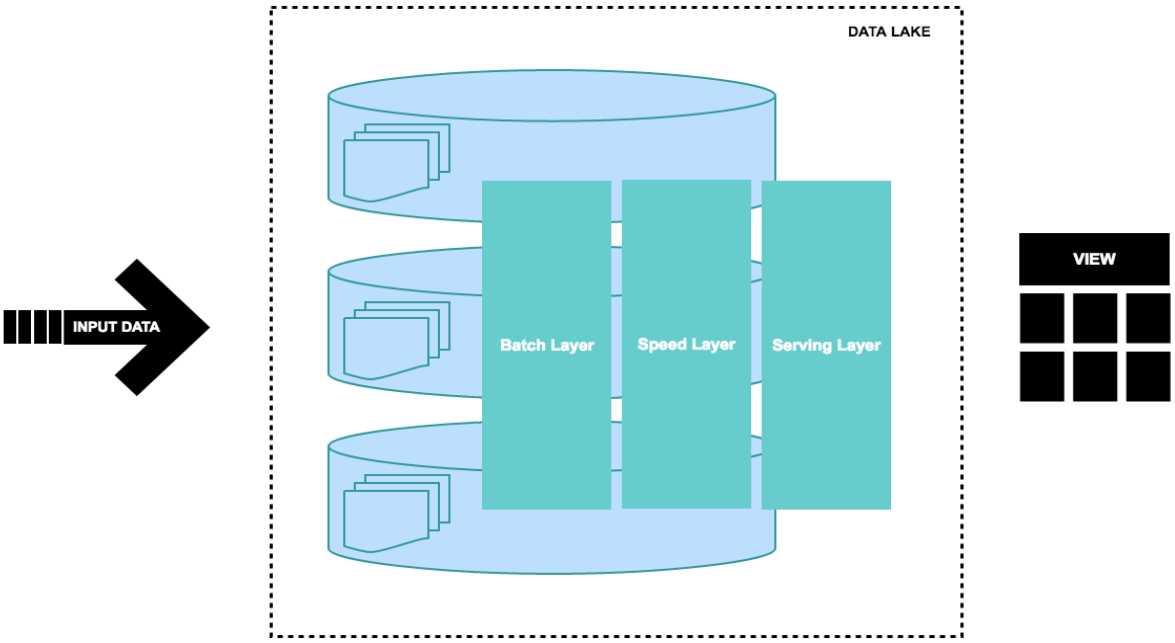


Рис. 18.4. Узагальнена Kappa – архітектура [3]

Деякі варіанти додатків для соціальних мереж, пристроїв, підключених до хмарної системи моніторингу, Інтернету речей (IoT), використовують оптимізовану версію архітектури Lambda, яка використовує служби швидкісного рівня в поєднанні з потоковим рівнем для обробки даних через Data Lake.

18.3. Kappa – архітектура

У 2014 році Джей Крепс почав обговорення, в якому вказав на деякі невідповідності архітектури Lambda, які в подальшому привели світ великих даних до іншої альтернативної архітектури – Kappa, яка використовувала менше ресурсів коду і була здатна добре працювати в певних корпоративних сценаріях. Архітектура Kappa може бути розгорнута для тих корпоративних моделей обробки даних, де:

- кілька подій або запитів даних заносяться в чергу для обробки в розподіленому сховищі файлової системи або в історії;
- порядок подій і запитів не визначений, платформи потокової обробки можуть взаємодіяти з базою даних в будь-який час;
- обробка терабайтів сховища потрібно для кожного вузла системи для підтримки реплікації.

Вищезазначені сценарії даних обробляються за допмогою платформи Apache Kafka, яка забезпечує швидкість обчислень, відмовостійкість і горизонтальну масштабованість. Це дозволяє поліпшити механізм управління потоками даних. Збалансоване управління потоковими процесорами і базами даних дозволяє програмам працювати відповідно до очікувань Kafka, зберігає впорядковані дані протягом більш тривалого часу і обслуговує аналогічні запити, пов'язуючи їх з відповідною позицією збереженого журналу в режимі реального часу.

LinkedIn і деякі інші додатки використовують цей різновид обробки великих даних і отримують вигоду від збереження великого обсягу даних для обслуговування тих запитів, які є простою копією один одного. Архітектура Kappa не може розглядатися як заміна архітектури Lambda, її слід розглядати як альтернативу, яка буде використовуватися в тих випадках, коли для забезпечення стандартної якості обслуговування не потрібно активна продуктивність пакетного рівня.

18.4. Переваги і недоліки Kappa-архітектури

Розглянемо переваги Kappa-архітектури. Архітектура Kappa може бути використана для розробки систем даних, які навчаються в режимі онлайн і тому не потребують пакетноого рівня:

- повторна обробка потрібна тільки при зміні коду;
- може використовуватися для горизонтально масштабованих систем;
- потрібно менше ресурсів, так як машинне навчання здійснюється в режимі реального часу.

Розглянемо недоліки Kappa-архітектури. Відсутність пакетного рівня може призвести до помилок при обробці даних або при оновленні бази даних, що вимагає наявності диспетчера винятків для повторної обробки даних. Вибір між архітектурою Lambda і Kappa здається компромісом. Якщо потрібна архітектура, більш надійна в оновленні Data Lake, а також ефективна в розробці моделей машинного навчання для надійного прогнозування нових подій, слід

використовувати архітектуру Lambda, оскільки вона використовує переваги пакетного рівня і швидкість шару, щоб забезпечити менше помилок.

Якщо потрібно розгорнути архітектуру великих даних з використанням менш дорогого обладнання і вимагати від неї ефективної обробки на основі унікальних подій, що відбуваються під час виконання, доцільно використовувати архітектуру Kappa для оброблення даних в реальному часі.

Висновок до лекції 18

Lambda – це архітектура обробки даних, яка використовує як потокову обробку, так і пакетну обробку, щоб отримати точний перегляд як «живих» даних, так і пакетних даних. Переваги архітектури Lambda:

1. пакетний рівень архітектури Lambda управляє історичними даними за допомогою відмовостійкого розподіленого сховища, яке забезпечує низьку ймовірність помилок, навіть якщо відбувається збій системи;

2. це хороший баланс швидкості і надійності;

3. відмовостійка і масштабована архітектура для обробки даних.

Недоліки архітектури Lambda:

1. накладні витрати кодування через використання комплексної обробки;

2. повторне оброблення кожного пакетного циклу, що не вигідно в певних сценаріях;

3. дані, змодельовані з використанням архітектури Lambda, важко перенести або реорганізувати.

Архітектура Kappa може бути використана для розробки інтелектуальних систем даних, які навчаються в режимі онлайн і тому не потребують пакетного рівня, коли повторна обробка потрібна тільки при зміні коду, може використовуватися для горизонтально масштабованих систем, де потрібно менше ресурсів, так як машинне навчання здійснюється в режимі реального часу.

Питання для закріплення

1. Опишіть Lambda – архітектуру оброблення великих даних.
2. Опишіть переваги і недоліки Lambda -архітектури.
3. Опишіть Kappa - архітектуру оброблення великих даних.
4. Опишіть переваги і недоліки Kappa-архітектури.

Список рекомендованої літератури

1. IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

2. IoT Analytics Platform // Електронний ресурс. Режим доступу: https://blog.codecentric.de/en/2016/07/iot-analytics-platform/

3. Akka // Електронний ресурс. Режим доступу: https://akka.io/

4. IoT-Analyse-Plattform: Floating Bus Data // Електронний ресурс. Режим доступу: https://www.youtube.com/watch?v=VYxc-3ZRRL4&ab_channel=codecentricAG

5. A brief introduction to two data processing architectures — Lambda and Kappa for Big Data // Електронний ресурс. Режим доступу: https://towardsdatascience.com/a-brief-introduction-to-two-data-processing-architectures-lambda-and-kappa-for-big-data-4f35c28005bb