Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСОБІВ ОБРОБКИ ВЕЛИКИХ НАБОРІВ ДАНИХ
RESEARCH AND SOFTWARE IMPLEMENTATION OF BIG DATA PROCESSING TOOLS

Виконав: студент 2 курсу, групи КНТ-214м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

МУРАВСЬКИЙ І.А.
(ПРІЗВИЩЕ та ініціали)
Керівник ПОЛЯКОВ М.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОРОТКИЙ О.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

МУРАВСЬКОГО Ігоря Андрійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація засобів обробки великих наборів даних. Research and Software Implementation of Big Data Processing Tools

керівник проєкту (роботи) д-р техн. наук, проф., ПОЛЯКОВ Михайло Олексійович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 30 " вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 04 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПОЛЯКОВ М.О., професор		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 26 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки		
4	Розробка структури програми.	3 тиждень	Розділ 3
5	Розробка програми.	4-7 тижні	Розділи 3,4
6	Тестування та експериментальне дослідження	8 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	9-10 тижні	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка) _____ Ігор МУРАВСЬКИЙ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Михайло ПОЛЯКОВ
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 97 с., 23 рис., 5 табл., 3 дод., 30 джерел.

PYTHON, PYSPARK, BIG DATA, INTERNET OF THINGS, РОЗРОБКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕЛИКІ НАБОРИ ДАНИХ, ПАРАЛЕЛІЗАЦІЯ

Об’єкт дослідження – процес обробки великих наборів даних.

Предмет дослідження – методи та засоби для обробки великих наборів даних.

Метою роботи є прискорення процесу обробки великих наборів даних, шляхом паралелізації.

Матеріали, методи та технічні засоби: мова програмування Python, середовище розробки PyCharm, бібліотеки та фреймворки PySpark та TensorFlow.

Результати. Розроблено програмне забезпечення для обробки великих наборів даних на основі мови програмування Python, середовища розробки PyCharm, бібліотеки PySpark та фреймворка TensorFlow.

Практична цінність роботи полягає у розробці програмного забезпечення, для обробки великих наборів даних.

Висновки. На основі досліджених переваг та недоліків засобів обробки великих наборів даних та виконаного проєктування нового рішення, розроблено новий засіб для обробки великих наборів даних. Здійснено тестування розробленого програмного забезпечення для обробки великих наборів даних.

Галузь використання – обробка великих та надмірних даних, онлайн-комерція, промислові виробництва.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 97 pages, 23 figures, 5 tables, 3 appendixes, 30 sources.

PYTHON, PYSPARK, BIG DATA, INTERNET OF THINGS, DEVELOPMENT, SOFTWARE, BIG DATA SETS, PARALLELIZATION

The object of research is the process of processing large data sets.

The subject of research is methods and tools for processing large data sets.

The goal of this paper is to speed up the processing of large data sets by parallelizing them.

Materials, methods, and technical tools: Python programming language, PyCharm development environment, PySpark and TensorFlow libraries and frameworks.

Results. Software for processing large data sets based on the Python programming language, the PyCharm development environment, the PySpark library, and the TensorFlow framework has been developed.

The practical value of the work lies in the development of software for processing large data sets.

Conclusions. Based on the advantages and disadvantages of processing large data sets and the design of a new solution, a new tool for processing large data sets has been developed. The developed software for processing large data sets was tested.

Scope of use: processing of large and redundant data, online commerce, industrial production.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BD – Big data;

IDE – integrated development environment;

IoT – Internet of Things;

RDD – Resilient Distributed Dataset;

ПЗ – програмне забезпечення.

ВСТУП

Великі дані (Big data – BD) стали актуальною темою сьогодні, тому що сучасне суспільство генерує інформацію в небачених раніше масштабах, з надмірними швидкістю і складністю, і це стрімке зростання обсягу даних має глибокий вплив практично на всі сфери: від науки і охорони здоров'я до фінансів, промисловості, державного управління, транспорту, соціальної поведінки, енергетичних систем. і глобальні комунікаційні мережі [1]. Розуміння того, чому великі дані настільки важливі зараз, вимагає розгляду кількох конвергентних факторів: технологічні зміни, що призвели до масових потоків даних, економічна та наукова цінності, приховані в цих даних, ризики та проблеми, пов'язані з необробленими або погано обробленими великими наборами даних, а також нагальна необхідність використання передових аналітичних та обчислювальних інструментів, здатних ефективно обробляти таку інформацію [1]. BD – це не лише технічне явище; це фундаментальний зсув у тому, як створюються знання, як приймаються рішення та як досягається конкурентна перевага в цифрову епоху. Саме тому вивчення і аналіз інструментів обробки BD став найважливішим пріоритетом, особливо з урахуванням того, що ці інструменти визначають здійсненність, надійність і продуктивність всіх процесів, керованих даними [1].

Обсяг даних, що накопичується щодня – надмірний. Завдяки мільярдам смартфонів, датчиків Інтернету речей (Internet of Things – IoT), пристроїв промислового моніторингу, систем дистанційного зондування, автономних машин, платформ соціальних мереж, наукових приладів і сервісів електронної комерції, що працюють одночасно, в даний час в світі щорічно генерується зеттабайт інформації [1]. Але актуальність полягає не тільки в обсязі. Різноманітність форматів-структуровані записи, неструктурований текст, мультимедійний вміст, потокові сигнали, машинні журнали, засоби відстеження здоров'я та геопросторові дані – все ускладнює. Традиційні системи обробки даних, розраховані на мегабайти або гігабайти, перевантажені

терабайтами, петабайтами, а тепер і ексабайтами потоків у реальному часі. Наприклад, великі наукові проєкти, такі як секвенування геному, космічні спостереження, моделювання клімату або експерименти з фізикою елементарних частинок, вимагають величезної кількості необроблених даних, які необхідно швидко обробляти, оскільки їх не можна зберігати нескінченно [1]. Промислові компанії постійно збирають телеметрію з обладнання для виявлення аномалій або оптимізації операцій. Банки аналізують мільйони фінансових операцій в секунду, щоб виявити шахрайство. Платформи соціальних медіа генерують нескінченні потоки даних про поведінку, які формують системи рекомендацій та рекламні стратегії. Так звані, розумні міста і сучасні транспортні системи, на основі IoT, покладаються на постійну аналітику схем руху, стану навколишнього середовища і споживання енергії. У всіх цих випадках обробка великих обсягів даних стає обов'язковою; вона необхідна для забезпечення того, щоб рішення були своєчасними, точними і ґрунтувалися на достовірній інформації [1].

Актуальність цього питання також обумовлена зростаючою залежністю підприємств і установ від прийняття рішень на основі даних. Компанії розуміють, що їх конкурентоспроможність залежить від того, наскільки швидко та розумно вони можуть отримувати інформацію з даних. Ті, хто успішно використовує аналітику великих даних, отримують стратегічні переваги: вони можуть раніше виявляти ринкові тенденції, точніше розуміти поведінку споживачів, більш ефективно оптимізувати виробничі процеси і знижувати операційні ризики [1]. Але жодна з цих переваг не може бути досягнута без складних інструментів, здатних обробляти масивні, розподілені та неоднорідні набори даних. У міру ускладнення джерел даних зростає потреба в платформах, що підтримують паралельну обробку, розподілені обчислення, відмовостійкість, високошвидкісну передачу даних і масштабовані архітектури зберігання. Без таких інструментів сучасні підприємства захлинулися б у власних даних, витрачаючи великі суми на зберігання і не отримуючи при цьому значущої інформації [1].

Наукові дослідження також значною мірою залежать від великих даних. Сучасна наука за замовчуванням стає обчислювальною, а це означає, що прориви часто відбуваються в результаті виявлення закономірностей у масивних наборах даних, а не в результаті дрібномасштабних експериментів або простих теоретичних моделей [2]. Наприклад, дослідження геноміки базуються на аналізі величезних масивів даних генетичних послідовностей, щоб зрозуміти причини захворювань або розробити індивідуальне лікування. Наукові дослідження клімату вимагають величезних обчислювальних потужностей для обробки даних про температуру, супутникові знімки, атмосферне моделювання та довгострокові екологічні характеристики. Фізичні експерименти на великих прискорювачах елементарних частинок дозволяють отримувати потоки необробленої інформації, яку необхідно відфільтровувати в режимі реального часу, оскільки лише мала частина подій має наукове значення. Астрономія та космологія залежать від безперервного потоку спостережень з телескопів та космічних обсерваторій. У кожному конкретному випадку здатність обробляти великі обсяги даних визначає темпи наукового прогресу. Без відповідних інструментів обробки вихідна інформація залишається марною, а потенційні знання стають недоступними [2].

Ще одна причина, чому великі дані є актуальними сьогодні, полягає в тому, що виникають нові типи ризиків, коли організації не можуть правильно обробляти або інтерпретувати масиви даних. При неправильному поводженні великі дані можуть приховувати помилки, спотворення, оманливі кореляції або операційні аномалії. Наприклад, у сфері кібербезпеки необхідно аналізувати величезні обсяги системних журналів для виявлення спроб вторгнення або незвичайної поведінки [2]. Несвоєчасне виявлення критичних аномалій може призвести до серйозних наслідків. У сфері охорони здоров'я необхідно ретельно обробляти величезні масиви медичних даних, щоб уникнути помилкових діагнозів або невірних прогнозів. У фінансовій сфері неправильний аналіз великих даних може призвести до помилкових сигналів про шахрайство або невиявлених системних ризиків [2]. У сфері державного управління

неправильне тлумачення даних може призвести до неефективної політики або нераціонального використання ресурсів. Великі дані самі по собі не представляють цінності; вони стають цінними лише при правильній обробці за допомогою відповідних інструментів. Ось чому важливо розуміти технології обробки великих обсягів даних: ці інструменти визначають точність, об'єктивність, прозорість та надійність систем, керованих даними [2].

Розвиток штучного інтелекту також робить BD все більш актуальними. Сучасні методи машинного навчання, особливо глибокого, значною мірою базуються на масивних наборах даних для досягнення високої продуктивності. Нейронні мережі, навчені розпізнаванню зображень, обробці природної мови, розпізнаванню мови, автономній навігації або прогнозному обслуговуванню, потребують мільйонів або мільярдів зразків даних [3]. Але для введення такої великої кількості інформації в моделі потрібні інструменти, здатні керувати розподіленим навчанням, паралельними обчисленнями, оптимізованим використанням пам'яті та високоефективними конвеєрами прийому даних. Поява великомасштабних моделей штучного інтелекту, таких як трансформаторні архітектури, що використовуються для обробки природного мовлення або мультимодальної обробки даних, було б неможливим без систем обробки великих даних. Такі інструменти, як розподілені файлові системи, бази даних в оперативній пам'яті, кластери паралельних обчислень, процесори на базі GPU і масштабовані хмарні інфраструктури, складають невидиму основу сучасного ШІ. Якщо організації не розуміють або не розробляють належним чином ці інструменти, системи ШІ стають повільними, неточними та нестабільними [3].

Актуальність BD також обумовлена потребою в аналітиці в режимі реального часу. Багато сучасних прикладних застосунків: автономні автомобілі, фінансові операції в Інтернеті, системи промислового моніторингу, телемедичні послуги, управління енергомережами та системи реагування на надзвичайні ситуації — вимагають негайної обробки даних [3]. Традиційні методи пакетної обробки даних занадто повільні для таких випадків

використання. Системи повинні миттєво виявляти закономірності, виявляти аномалії протягом мілісекунд, реагувати на несподівані події та динамічно адаптуватися. Для цього потрібні передові засоби обробки, здатні виконувати потокову аналітику, розподілені обчислення та конвеєри передачі даних з високою пропускною здатністю. Оскільки прийняття рішень у режимі реального часу займає центральне місце в сучасній інфраструктурі, інструменти обробки BD стають критичними [3].

Зростаюча складність екосистем даних ще раз показує, чому BD є актуальною темою. Дані більше не зберігаються в одному місці, а розподіляються на хмарних платформах, периферійних пристроях, гібридних інфраструктурах, приватних центрах обробки даних та віддалених вузлах зберігання даних. Зв'язки між цими середовищами створюють нові проблеми: синхронізація даних, управління затримками, захист конфіденційності, безпечна передача та ефективне стиснення [3]. Щоб впоратися з цією складністю, організації повинні використовувати інструменти обробки великих обсягів даних, які підтримують розподілені системи, відмовостійку архітектуру та ефективний розподіл ресурсів. Такі технології, як Apache Hadoop, Spark, Flink, Kafka, Storm, Dask, Ray та хмарні платформи обробки даних, забезпечують паралельну обробку та масштабовану аналітику. Розуміння того, які інструменти використовувати, як вони працюють і як їх інтегрувати, визначає, чи зможе система, керована даними, працювати надійно. Актуальність цього питання обумовлена тим фактом, що без належних інструментів організації втрачають контроль над своїми екосистемами даних і не можуть гарантувати якість, продуктивність і безпеку даних [4].

Однією з найважливіших причин аналізу інструментів обробки великих даних є те, що вибір інструментів суттєво впливає на вартість обчислень. Зберігання та обробка масивних наборів даних коштує дорого. Постачальники хмарних технологій стягують плату за зберігання, пропускну здатність та обчислення [4]. Неефективні шляхи передачі даних можуть призвести до непотрібного дублювання, надмірних операцій вводу-виводу та збільшення

часу обробки. Неправильно підібрані інструменти можуть вимагати дорогої модернізації обладнання або надмірного споживання енергії. Масштабовані та оптимізовані інструменти обробки великих даних знижують експлуатаційні витрати за рахунок розумного розподілу робочого навантаження, кешування часто використовуваних даних, оптимізації використання пам'яті та ефективного розпаралелювання обчислень. Для організацій з обмеженим бюджетом, особливо для науково-дослідних установ або малих та середніх підприємств, інструменти обробки великих даних мають значний економічний вплив. Розуміння цих інструментів – це не лише технічна необхідність, а й фінансова стратегія [4].

Іншим важливим аспектом є управління даними та їх конфіденційність. У міру зростання обсягу даних зростають і ризики, пов'язані з порушенням конфіденційності, несанкціонованим доступом і неправомірним використанням особистої інформації. Низка галузей: охорона здоров'я, фінанси, освіта, державне управління – перебувають під суворим регулюванням. Інструменти обробки BD повинні підтримувати безпечну обробку, анонімізацію, шифрування, контроль доступу та контрольні журнали. Розуміння того, які інструменти забезпечують надійні функції безпеки, важливо для створення сумісних та надійних систем. Актуальність BD безпосередньо пов'язана з необхідністю етичного та юридично відповідального поводження з інформацією [4].

З точки зору суспільства в цілому, великі дані є актуальними, оскільки вони визначають державну політику, економічні стратегії та глобальні комунікаційні системи. Уряди покладаються на величезні масиви даних для управління охороною здоров'я, транспортом, національною безпекою, освітою та соціальним забезпеченням. Наприклад, точне прогнозування пандемії залежить від аналізу великих масивів даних про інфекції, особливості мобільності, генетичні мутації та соціальну поведінку. Економічне планування ґрунтується на аналізі тенденцій в області зайнятості, демографічних зрушень, моделей споживання і показників промислового виробництва [4]. Системи

реагування на катастрофи залежать від даних датчиків у режимі реального часу, супутникових знімків та сигналів зв'язку. Інструменти, що використовуються для обробки цих наборів даних, визначають, наскільки швидко та точно можуть діяти уряди. Недоліки в обробці BD можуть призвести до неефективної політики, запізнілого реагування або прийняття невірних рішень [4].

Нарешті, необхідний аналіз інструментів обробки BD, оскільки технологічний ландшафт швидко розвивається. З'являються нові платформи, старі – застарівають, а архітектурні підходи змінюються в міру розвитку апаратних, мережевих та обчислювальних парадигм. Інструментів, які були домінуючими десять років тому, зараз недостатньо для поточних навантажень. Наприклад, традиційна пакетна обробка даних Hadoop замінюється або доповнюється технологіями обробки даних в оперативній пам'яті і потокової передачі даних. Архітектури на основі графічних процесорів (graphics processing unit – GPU) і тензорних блоків обробки (tensor processing unit – TPU) змінюють формат обробки даних, керованої штучним інтелектом. Передові обчислення скорочують час очікування, виконуючи аналітику поблизу джерела даних. Квантові обчислення починають впливати на дискусії щодо майбутніх можливостей обробки даних. Таким чином, розуміння інструментів є безперервним процесом, оскільки кожне покоління технологій пропонує різні характеристики продуктивності, межі масштабованості, моделі програмування та структуру витрат. Актуальність виникає тому, що організації повинні постійно оцінювати, чи залишаються обрані ними інструменти актуальними та ефективними у швидко мінливому середовищі [4].

Підводячи підсумок, великі дані є актуальною темою сьогодні, оскільки світ переступив поріг, коли інформація існує в таких масштабах і в таких формах, які вимагають нових методів обробки та інтерпретації [4]. Важливість аналізу інструментів обробки великих обсягів даних полягає в тому, що ці інструменти дозволяють організаціям, науковцям і суспільству перетворювати необроблені дані в значущі знання, приймати своєчасні і точні рішення, підтримувати операційну ефективність, захищати конфіденційність, знижувати

ризики і залишатися конкурентоспроможними в швидко мінливому цифровому ландшафті. Без розуміння цих інструментів величезний потенціал великих даних залишається нереалізованим, а пов'язані з ними проблеми: складність, вартість, ризик та невизначеність – стають непереборними [4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Що таке BD і чим вона відрізняється від звичайних даних

BD – це надзвичайно великі, складні та швидко зростаючі масиви інформації, які неможливо ефективно збирати, зберігати, керувати ними чи аналізувати їх за допомогою традиційних методів обробки даних або стандартних баз даних [5]. Концепція не залежить від конкретного числового порогу, натомість великі дані описують набори даних, масштаб, структуру, швидкість або мінливість яких перевищує технічні можливості звичайних інструментів, які спочатку були розроблені для обробки меншої, добре структурованої та повільно мінливої інформації. Великі дані часто включають інформацію, що постійно генерується цифровими пристроями, датчиками, онлайн-системами та взаємодією людей, і можуть відображатися у найрізноманітніших форматах, таких як текст, зображення, відеопотоки, Числові журнали, геолокація, активність у соціальних мережах або машинна телеметрія. Великі дані принципово відрізняються від звичайних не тільки своїм розміром, але і поєднанням обсягу, швидкості, різноманітності, мінливості і складності, що створює нові завдання для зберігання, обробки та аналізу [5].

Звичайні дані, як правило, розміщуються в реляційних базах даних або структурованих таблицях із заздалегідь визначеними рядками, стовпцями та взаємозв'язками. Зазвичай вони створюються в невеликих масштабах і зберігаються в системах, до яких можна отримати доступ за допомогою традиційних методів, таких як SQL-запити. Наприклад, записи транзакцій малого бізнесу, дані пацієнтів у місцевій клініці або відповіді на опитування, що зберігаються в електронних таблицях, є прикладами звичайних даних. Цими наборами даних можна керувати на одному комп'ютері або в невеликому серверному середовищі, а обчислювальні ресурси, необхідні для виконання запитів, агрегування або аналізу, невеликі [5]. Традиційні інструменти аналізу

даних можуть обробляти ці набори даних, надаючи аналітичну інформацію без проблем із продуктивністю чи масштабованістю [6].

BD, навпаки, перевищують фізичні та логічні можливості таких систем. Перша важлива відмінність полягає в масштабі. BD часто містять терабайти, петабайти або навіть ексабайти інформації, розподіленої по безлічі серверів або хмарних ресурсів. Такий обсяг не дозволяє зберігати дані в єдиній базі даних або обробляти їх на одному комп'ютері. Натомість використовуються розподілені системи зберігання даних, такі як розподілена файлова система Hadoop, хмарне сховище об'єктів або паралельні бази даних, що дозволяють розподіляти дані на декількох комп'ютерах [6]. Для обробки таких великих обсягів даних потрібні розподілені обчислювальні платформи, які розбивають завдання на менші частини і працюють над ними одночасно. Ця архітектура принципово відрізняється від традиційних централізованих систем, оскільки робить BD не просто великим обсягом даних, а іншим типом даних, що вимагає іншої обчислювальної парадигми [6].

Другою важливою відмінністю є швидкість. Великі дані часто генеруються з високою швидкістю: тисячі або мільйони подій в секунду-за допомогою датчиків, фінансових операцій, соціальних мереж, промислового обладнання або вебвзаємодій. Обсяг звичайних даних поступово збільшується, і їх можна обробляти пакетами без жорстких часових обмежень [6]. BD надходять безперервно, а це означає, що системи повинні отримувати, зберігати та аналізувати їх у режимі реального часу або майже в режимі реального часу. Для такого швидкого надходження потрібні потокові платформи, такі як Apache Kafka або Spark Streaming, які можуть обробляти безперервні потоки даних. Необхідність швидкого реагування створює проблеми при роботі з неповними, зашумленими або частково обробленими даними, з якими традиційні інструменти не можуть ефективно впоратися [6].

Третя відмінність – різноманітність. Звичайні дані, як правило, добре структуровані та послідовні. Великі дані включають широкий спектр форматів: структуровані дані, такі як таблиці, напівструктуровані дані, такі як JSON або

XML, і неструктуровані дані, такі як текстові документи, відео, аудіосигнали або показання датчиків [6]. Таке різноманіття ускладнює зберігання та аналіз, оскільки жодна єдина схема не може описати всі типи даних. Для масштабної інтерпретації неструктурованих даних часто потрібні методи машинного навчання та обробки природнього мовлення. Це фундаментальний перехід від традиційного управління даними, де схеми та формати були стабільними, передбачуваними та контрольованими [6].

Інша відмінність полягає в мінливості і неузгодженості. У середовищі BD часто спостерігаються непередбачувані стрибки обсягу даних або раптові зміни в структурі та змісті даних [7]. Наприклад, активність у соціальних мережах може різко зрости під час виходу останніх новин, або дані датчиків можуть коливатися через зміни навколишнього середовища або операційної діяльності. Традиційні бази даних вимагають постійного та передбачуваного робочого навантаження, але системи BD повинні динамічно адаптуватися до мінливих умов. Така непередбачуваність створює додаткове навантаження на інфраструктуру, вимагаючи гнучкого масштабування, автоматизованого розподілу ресурсів і відмовостійкості [7].

Складність також суттєво відрізняє BD. Екосистеми великих даних включають багато рівнів взаємодіючих компонентів: розподілене сховище, шляхи обробки даних, менеджери кластерів, потокові процесори, сховища даних, послуги індексації та платформи машинного навчання. Управління цією екосистемою вимагає нових навичок та інструментів, які виходять за рамки традиційного адміністрування баз даних [7]. Проєктування архітектур BD передбачає розуміння розподілених алгоритмів, вузьких місць у мережі, відмовостійкості, паралельної обробки та поведінки великомасштабної системи. Користувачі повинні враховувати такі проблеми, як реплікація даних, моделі послідовності, балансування навантаження та розподілене виконання запитів – проблеми, які рідко виникають у звичайних системах обробки даних [7].

BD відрізняються ще й тим, що вони часто містять шум, надмірність і неповну інформацію. Звичайні дані, як правило, перед збереженням проходять ретельну обробку, контроль і валідацію. BD можуть бути необробленими, безладними та повними помилок, що вимагає таких методів попередньої обробки, як: очищення даних, фільтрація, дедуплікація, нормалізація та вилучення ознак. Моделі машинного навчання повинні бути навчені відрізняти значущі моделі від нерелевантної або оманливої інформації. Наявність шуму та складності робить аналіз великих даних не лише обчислювальним завданням, а й методологічним [7].

Інша відмінність стосується мети та результатів, очікуваних від аналізу великих даних. Регулярний аналіз даних часто дає відповіді на конкретні, чітко визначені запитання, використовуючи описову статистику або прості запити. Аналіз великих даних більше орієнтований на дослідження й прогнозування та спрямований на виявлення прихованих закономірностей, кореляцій або аномалій, які неможливо виявити в невеликих наборах даних [8]. Для цього часто потрібні передові методи, такі як глибоке навчання, графічний аналіз, моделювання часових рядів або обробка природної мови. Аналіз BD дозволяє приймати рішення в режимі реального часу та отримувати стратегічну інформацію в таких сферах, як охорона здоров'я, фінанси, маркетинг, транспорт та промислова автоматизація [8].

BD також вимагають нових підходів до управління даними, забезпечення конфіденційності та безпеки. Звичайні системи обробки даних, як правило, керують контрольованими та обмеженими наборами даних із встановленими правилами доступу. Системи великих даних повинні захищати масивні та різноманітні набори даних, часто розподілені по декількох місцях і керовані різними зацікавленими сторонами [8]. Правові норми, такі як закони про конфіденційність та механізми захисту даних, мають суворі вимоги щодо способу зберігання, обробки та спільного використання великомасштабних наборів даних. Чим більші та різноманітніші дані, тим більший ризик порушення конфіденційності, розкриття особистих даних або зловживання

даними. Таким чином, системи BD повинні інтегрувати шифрування, анонімізацію, контроль доступу та аудит на набагато глибшому рівні, ніж традиційні системи [8].

Нарешті, BD відрізняються від звичайних тим, що вони часто використовуються в системах, що взаємодіють з фізичним світом. Аналітика в режимі реального часу в автономних транспортних засобах, інтелектуальних мережах, промислових роботах та моніторингу охорони здоров'я залежить від негайної та точної інтерпретації величезних потоків даних [8]. Помилки можуть призвести до загрози безпеці, фінансових втрат або зупинки роботи. Така оперативна терміновість створює ще один рівень складності і робить великі дані набагато важливішими, ніж звичайні дані, що зберігаються для ведення обліку або періодичної звітності [8].

Таким чином, BD не просто більші за звичайні дані; вони принципово відрізняються за масштабом, швидкістю, структурою, складністю та призначенням. Для отримання значущої інформації та підтримки прийняття рішень у режимі реального часу потрібні розподілені системи, вдосконалена аналітика та нові обчислювальні моделі [8].

1.2 Програмні засоби для обробки великих наборів даних

Обробка BD спирається на спеціалізовані програмні засоби, інтегровані середовища розробки та великомасштабні платформи, призначені для зберігання, управління, розповсюдження та аналізу надзвичайно великих та неоднорідних наборів даних. Ці інструменти працюють інакше, ніж традиційне програмне забезпечення, оскільки вони повинні обробляти великі обсяги даних, високошвидкісні потоки, різні формати, розподілене сховище та паралельні обчислення [9]. Системи BD зазвичай працюють на кластерах комп'ютерів, часто в хмарі, де дані розподіляються та обробляються одночасно багатьма вузлами. Розуміння основних інструментів та принципів їхньої роботи допомагає прояснити технологічні основи сучасних систем і застосунків,

керованих даними. Багато інструментів поділяються на такі категорії, як платформи розподіленого зберігання, пакетної обробки, потокові двигуни, бібліотеки розподіленого машинного навчання, рівні оркестрації та хмарні платформи обробки BD [9].

Одним з основних інструментів для обробки BD є екосистема Hadoop, яка представила ключові концепції розподіленого зберігання та паралельних обчислень. Основний компонент Hadoop – розподілена файлова система Hadoop, працює, розділяючи великі набори даних на блоки та розподіляючи їх по декількох серверах, кожен з яких зберігає надлишкові копії для забезпечення відмовостійкості. Це запобігає втраті даних у разі збою сервера і дозволяє виконувати паралельне читання блоків даних [9]. Ключовим принципом є локальність даних: обчислення переносяться на сервер, де знаходиться блок даних, що знижує навантаження на мережу. MapReduce, ще одна основна частина Hadoop, працює, розділяючи обчислення на два етапи. На етапі зіставлення обробляються вхідні дані і створюються пари ключ-значення, в той час як на етапі скорочення ці пари об'єднуються або перетворюються для отримання кінцевих результатів. Цей підхід підтримує широкомасштабну пакетну обробку наборів даних, які занадто великі, щоб помістится на одному комп'ютері. Хоча MapReduce ефективний для простих завдань, таких як підрахунок, сортування або групування, він повільний для ітеративних алгоритмів, що вимагає розробки нових інструментів [9].

Apache Spark став однією з найбільш широко використовуваних платформ для обробки великих обсягів даних, оскільки він пропонує обчислення в пам'яті, що робить його значно швидшим, ніж MapReduce на основі диска, для багатьох робочих навантажень. Архітектура Spark базується на стійких розподілених наборах даних, які є незмінними, відмовостійкими наборами елементів, розподіленими по вузлах кластера [9]. Spark зберігає дані в пам'яті, коли це можливо, що дозволяє швидко виконувати ітеративні алгоритми, такі як оптимізація машинного навчання, графічний аналіз та аналітика в реальному часі. Spark підтримує кілька обчислювальних моделей в

межах однієї платформи, включаючи пакетну обробку, SQL-подібні запити, шляхи машинного навчання, обробку графіків та структуровану потокову передачу. Застосунок Spark координується драйвером, який створює логічний план виконання, та виконавцями, що працюють на вузлах кластера, які виконують паралельні обчислення. Spark інтегрується з багатьма системами зберігання даних, такими як HDFS, хмарні сховища об'єктів, бази даних NoSQL та системи розподіленої пам'яті [9].

Apache Flink – це ще один сучасний фреймворк для потокової та пакетної обробки, який відноситься до потокової передачі як до першокласного засобу обслуговування. Хоча Spark спочатку розглядав потокове передавання як мікропакетну обробку, Flink підтримує обробку від події до події з надзвичайно низькою затримкою [10]. Flink розподіляє обчислення по вузлах, використовуючи спрямовані плани виконання ациклічних графіків, а його потоковий механізм відстеження стану відстежує змінні стани, такі як лічильники, ковзаючі середні показники або розпізнавання образів. Механізм визначення контрольних точок Flink забезпечує відмовостійкість: він періодично зберігає стан операторів, щоб при відновленні можна було перезапустити обчислення з останнього узгодженого стану після збою. Flink широко використовується в системах моніторингу в режимі реального часу, виявлення шахрайства, аналітики IoT та промислової телеметрії, де дані повинні оброблятися протягом мілісекунд [10].

Apache Kafka виконує іншу роль в екосистемі великих даних, слугуючи платформою для розподіленого обміну повідомленнями та потокової передачі даних. Kafka не є обчислювальним механізмом; скоріше, він ефективно переміщує дані між системами. Kafka організовує дані за темами та розділами, дозволяючи кільком виробникам та споживачам паралельно записувати та читати повідомлення. Kafka зберігає дані на диску відмовостійким чином і реплікує розділи на серверах [10]. Така конструкція дозволяє Kafka обробляти мільйони повідомлень в секунду і виконувати функцію надійного буфера між виробниками даних, такими як датчики або застосунки, і механізмами

подальшої обробки, такими як Spark або Flink. Kafka незамінний в архітектурах великих даних реального часу, оскільки він роз'єднує системи, забезпечує надійну доставку повідомлень і надає можливості відтворення, дозволяючи аналітичним інструментам повторно обробляти історичні потоки даних [10].

Для розробників, що працюють з розподіленою обробкою даних, Apache Beam надає уніфіковану модель програмування, що дозволяє створювати конвеєри обробки даних, які можуть виконуватися на різних виконавчих механізмах, таких як Spark, Flink або Google Cloud Dataflow. Конвеєри Beam складаються з перетворень, що застосовуються до наборів даних, і виконуються розподіленим чином з використанням обраної серверної частини [10]. Таке відділення логіки від середовища виконання забезпечує гнучкість при виборі або перемиканні механізмів обробки без переписування коду [10].

Машинне навчання в масштабі великих даних також вимагає спеціалізованих інструментів. Apache Mahout була ранньою бібліотекою для машинного навчання в MapReduce, але через обмеження MapReduce щодо ітеративної обробки з'явилися більш сучасні рішення. MLlib, що входить до складу Apache Spark, надає масштабовані реалізації алгоритмів машинного навчання, які працюють з розподіленими наборами даних і використовують переваги архітектури Spark в оперативній пам'яті. MLlib включає алгоритми класифікації, кластеризації, регресії, спільної фільтрації та перетворення об'єктів [10]. Він також підтримує потоки, які поєднують етапи підготовки даних, навчання моделі та оцінки. Розподілені платформи глибокого навчання, такі як TensorFlow та PyTorch, можуть інтегруватися з екосистемами великих даних, використовуючи такі роз'єми, як TensorFlowOnSpark або Horovod. Ці інструменти дозволяють навчати великі нейронні мережі на декількох графічних процесорах або серверах, а сервери параметрів або універсальні комунікаційні шаблони координують оновлення ваги [10].

Розподілене сховище, яке виходить за рамки Hadoop, також відіграє важливу роль. Бази даних NoSQL, такі як Apache Cassandra, MongoDB та HBase, призначені для зберігання напівструктурованих або неструктурованих даних у

масштабі багатьох вузлів. Кассандра наголошує на високій доступності та відмовостійкості, розподіляючи дані по кластерах за допомогою послідовного хешування та підтримуючи швидкий запис. MongoDB зберігає гнучкі документи у форматі JSON і масштабується горизонтально за допомогою механізмів сегментації. HBase, побудований поверх HDFS, оптимізований для зберігання даних із широкими стовпцями та забезпечує швидкий довільний доступ для читання/запису, що робить його придатним для розріджених наборів даних або часових рядів даних [10].

Apache Hive та Apache Impala полегшують виконання SQL-запитів до великих даних. Hive перетворює SQL-запити в завдання MapReduce, Spark або Tez, дозволяючи аналітикам даних працювати з великими наборами даних, використовуючи знайомий синтаксис запитів [10]. З іншого боку, Impala виконує розподілені запити безпосередньо в пам'яті, забезпечуючи інтерактивну продуктивність, подібну до традиційних реляційних баз даних, але поверх розподіленого сховища. Presto (нині Trino) – це ще один високошвидкісний механізм розподілених запитів SQL, призначений для виконання інтерактивної аналітики великих наборів даних, що зберігаються у багатьох джерелах. Presto працює, розбиваючи запити на етапи та виконуючи їх паралельно на всіх вузлах [10].

Координація та управління кластерами мають вирішальне значення при обробці великих обсягів даних. Apache YARN, Kubernetes і Apache Mesos є основними платформами, які керують обчислювальними ресурсами в кластерах. YARN виділяє контейнери з процесором та пам'яттю для програм Hadoop та Spark. Kubernetes стає все більш популярним, оскільки організовує контейнерні програми в хмарі або локальних кластерах і підтримує автоматичне масштабування, відмову та планування ресурсів. Багато платформ обробки великих даних, такі як Spark, Flink та Kafka, зараз працюють на Kubernetes. Це забезпечує динамічне масштабування: при збільшенні навантаження можна додавати додаткових працівників і видаляти їх, коли вони не потрібні [10].

Airflow, Prefect та Dagster організовують робочі процеси у великих потоках даних. Ці інструменти планують і координують послідовність завдань, таких як прийом даних, попередня обробка, навчання моделі та створення звітів. Airflow використовує спрямовані ациклічні графіки для представлення робочих процесів та виконання завдань на різних обчислювальних серверах. Ці інструменти оркестрації забезпечують надійну роботу конвеєрів, обробку повторних спроб, управління залежностями і моніторинг [10].

Хмарні провайдери розробили інтегровані платформи для обробки YARN, які багато в чому спрощують Налаштування кластерів вручну. Наприклад, Amazon EMR використовує Hadoop, Spark, Hive, Flink та інші інструменти в керованому середовищі, де підготовка, масштабування та моніторинг автоматизовані. AWS Glue забезпечує безсерверний прийом даних, каталогізацію та обробку ETL. Google Cloud Dataflow використовує потоки Apache Beam і автоматично оптимізує використання ресурсів [10]. BigQuery – це повністю керована аналітична база даних, здатна запитувати петабайти даних за допомогою розподіленої стовпчастої системи зберігання та масово паралельного виконання. Azure пропонує такі послуги, як HDInsight, Synapse Analytics та Databricks в Azure, забезпечуючи інтеграцію розподілених обчислень, SQL-аналітики та машинного навчання. Ці хмарні платформи базуються на великомасштабних розподілених файлових системах, віртуалізованих кластерах та автоматичному масштабуванні, що робить обробку великих обсягів даних доступною для команд, які не мають глибоких знань щодо управління кластерами [9], [10].

Databricks, створений на базі Apache Spark, забезпечує середовище для співпраці, що поєднує процеси розробки даних, Data science та машинного навчання. Він поєднує в собі блокноти, управління кластерами та оптимізований час виконання Spark. Databricks використовує концепцію під назвою Delta Lake, яка забезпечує транзакції ACID в озерах даних, забезпечуючи надійне оновлення даних, застосування схем і управління версіями. Delta Lake та подібні технології, такі як Apache Iceberg та Hudi,

підвищують надійність великомасштабних середовищ зберігання, які традиційно страждають від відсутності гарантій виконання транзакцій [9], [10].

Інтегровані середовища розробки також підтримують обробку великих обсягів даних. Такі інструменти, як Jupyter Notebook, Zeppelin і Databricks notebooks, забезпечують інтерактивну розробку конвеєрів передачі даних, візуальне дослідження і поетапне тестування. Вони підключаються до серверної частини кластера і дозволяють розробникам виконувати розподілені обчислення в інтерактивному середовищі. PyCharm, IntelliJ та Visual Studio Code також підтримують розробку великих даних за допомогою плагінів для Spark, Hadoop та хмарних сервісів. Ці IDE полегшують налагодження, організацію коду, управління залежностями та інтеграцію з розподіленими системами [9], [10].

Платформи обробки графіків, такі як Apache Giraph та GraphX (бібліотека графіків Spark), дозволяють розподілені обчислення на великих графіках, таких як соціальні мережі, біологічні взаємодії або системи рекомендацій. Вони розбивають графіки на вузли та запускають ітеративні алгоритми, які поширюють інформацію по краях, наприклад, PageRank, підключені компоненти або виявлення спільноти [9], [10].

Нарешті, обробка BD включає інструменти моніторингу та реєстрації, такі як Prometheus, Grafana, ELK stack (Elasticsearch, Logstash, Kibana), а також хмарні служби моніторингу. Ці інструменти відстежують працездатність кластера, затримку, пропускну здатність, використання сховища та збої, дозволяючи адміністраторам підтримувати ефективні потоки обробки даних [10].

Таким чином, обробка BD спирається на потужну екосистему розподілених систем зберігання даних, платформ пакетної та потокової обробки, движків SQL, бібліотек машинного навчання, інструментів оркестрації, хмарних платформ і середовищ розробки. Ці інструменти працюють шляхом розділення наборів даних на частини та розподілу їх по безлічі машин, паралельної обробки, координації обчислень, забезпечення

надійності та надання інтерфейсів для аналізу та розробки додатків. У сукупності вони дозволяють організаціям управляти масивними, складними і швидко мінливими даними і отримувати з них користь, з якою традиційні системи впоратися не в змозі [10].

У таблиці 1.1 наведемо порівняння фреймворків та екосистем для роботи з BD.

Таблиця 1.1 – Порівняння фреймворків та екосистем для обробки BD

	Hadoop (MapReduce)	Apache Spark	Apache Flink
Модель первинної обробки	Пакетна обробка	Єдина пакетна і мікропакетна потокова передача даних	Потокове передавання в реальному часі та пакетна обробка
Основні переваги	Висока надійність; обробляє величезні обсяги даних; проста масштабована архітектура	Надзвичайно швидка обробка даних; обчислення в оперативній пам'яті; багата екосистема	Краща продуктивність у режимі реального часу; керована подіями
Слабкість	Повільність при виконанні ітеративних завдань; висока затримка; операції з великим обсягом сховища	Високі вимоги до пам'яті; мікропакетна обробка даних може обмежувати точність у режимі реального часу	Більш складна екосистема
Типові варіанти використання	ETL, масштабні пакетні завдання, обробка журналів	Машинне навчання, аналітика, обробка графіків, мікропакетне потокове передавання даних	Аналітика в реальному часі, обробка подій, безперервні конвеєри
Тип затримки	Висока затримка (хвилини / години)	Низька / середня затримка (секунди)	Дуже низька затримка (мілісекунди)
Масштабованість	Дуже висока	Дуже висока	Дуже висока
Мова програмування	Java, Python, R та інші за допомогою потокової передачі даних Hadoop	Scala, Java, Python, R.	Java, Scala, Python
Складність моделі	Проста модель програмування	Помірна складність; багатий набір API	Більш висока складність, потокова підтримка
Розгортання	Локальна або хмарна	Локальний або хмарний	Локальна або хмарна
Найкраще для	Застарілі робочі навантаження з великими обсягами даних	Швидка аналітика, загальні робочі процеси з великими даними	Критично важливе потокове передавання в режимі реального часу

З результатів порівняння помітно, що Spark є найбільш привабливою надбудовою для подальшої роботи над застосунком для обробки BD.

1.2.1 Сервіс Tableau

Tableau – це провідний інструмент візуалізації даних, який полегшує створення інтерактивних візуалізацій [11], [12]. Його зручний інтерфейс дозволяє нетехнічним користувачам візуалізувати дані і отримувати інсайти зі складних наборів даних (рис. 1.1).

Ключові функції:

– інтерфейс drag-and-drop для створення візуалізацій;

– підключення до баз даних NoSQL, хмарних платформ та джерел даних, таких як SQL Server;

– підтримка великі обсяги аналітичних даних з плавною масштабованістю;

– пропонує веб-панелі для спільної роботи команди.

Маркетингові команди використовують Tableau для створення панелей управління для аналізу ефективності кампаній [12].

Привабливим цей продукт більшість користувачів вважають, через інтуїтивний дизайн Tableau робить його популярним інструментом для аналітики в різних галузях.

Здатність Tableau об'єднувати дані з декількох систем (наприклад, CRM, ERP і хмарне сховище) робить його ідеальним для створення комплексних візуалізацій. Функція інсайтів на основі штучного інтелекту покращує предиктивну аналітику, допомагаючи бізнесу прогнозувати тенденції [13].

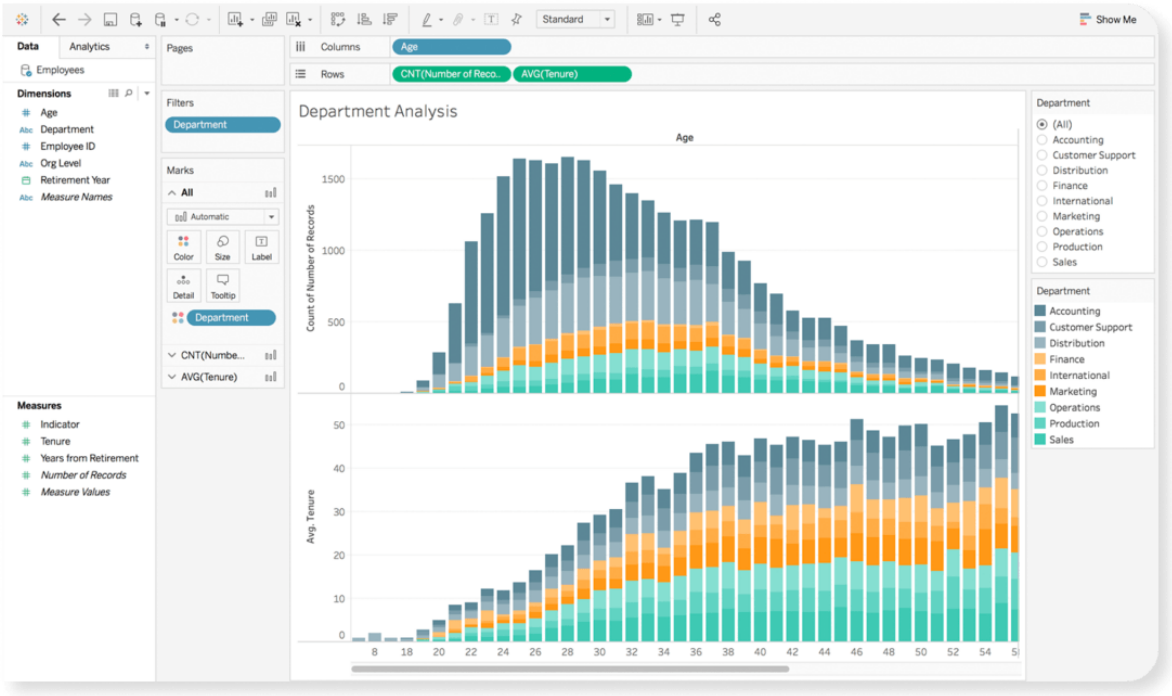


Рисунок 1.1 – Приклад графічного інтерфейсу Tableau з офіційного сайту [11]

1.2.2 Сервіс Microsoft Power BI

Microsoft Power BI – це інструмент аналітики даних, який перевершує візуалізацію даних та бізнес-аналітику [14] (рис. 1.2). Він призначений для допомоги користувачам в об'єднанні даних з різних джерел і створенні корисних інсайтів [15].

Ключові функції:

– інтуїтивні інструменти візуалізації для створення інтерактивних звітів;

– інтегрується з системами даних, такими як Azure, Excel та SQL Server;

– підтримує прогнозну аналітику для прогнозування тенденцій;

– обробляє великі набори даних з хмарною масштабованістю.

Роздрібні компанії використовують Power BI для аналізу даних для оптимізації запасів на основі даних часових рядів [15].

Привабливим цей продукт більшість користувачів вважають, через інтеграцію Power BI з екосистемою Microsoft, що робить його найкращим інструментом для певного прошарку підприємств [16].

Функція запитів на природній мові Power BI дозволяє користувачам задавати питання про свої дані простою англійською мовою, що робить його доступним для нетехнічних команд. Його можливості управління даними забезпечують відповідність стандартам корпоративного управління [16].

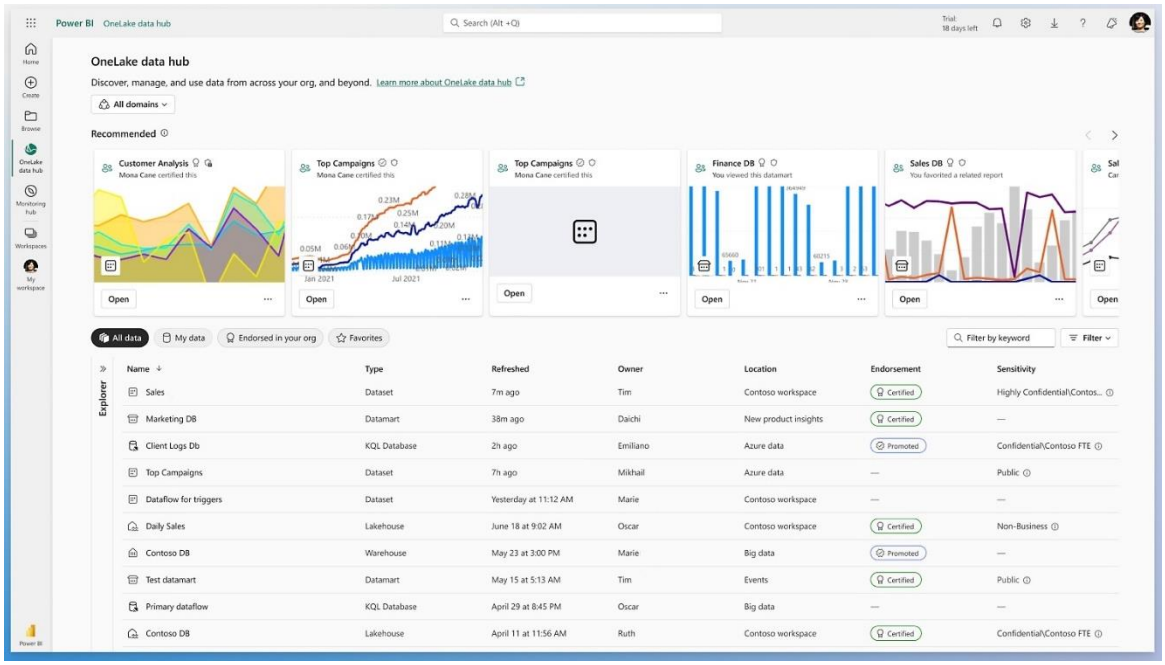


Рисунок 1.2 – Приклад графічного інтерфейсу Microsoft Power BI з офіційного сайту [14]

1.2.3 Інструмент RapidMiner

RapidMiner – це інструмент науки про дані, який інтегрує видобуток даних, машинне навчання та аналіз даних в єдину платформу. Він ідеально підходить для користувачів, які бажають аналізувати дані без широкого програмування [17] (рис. 1.3).

Ключові функції:

– візуальний конструктор робочих процесів для створення

інструментів аналізу даних;

– підтримує неструктуровані дані та аналіз часових рядів;

– інтегрується з мовами програмування, такими як R і Python, для просунутої аналітики;

– пропонує можливості дослідження даних для виявлення закономірностей.

Виробничі компанії використовують RapidMiner для прогнозної аналітики в обслуговуванні обладнання [18].

Привабливим цей продукт більшість користувачів вважають, через підхід RapidMiner без програмування робить його доступним для завдань аналітики даних [18].

Велика бібліотека готових моделей RapidMiner прискорює процеси інтелектуального аналізу даних, а інтеграція з хмарними платформами підтримує великомасштабний аналіз даних. Це особливо корисно для досліджень на основі даних в академічному середовищі та промисловості [18].

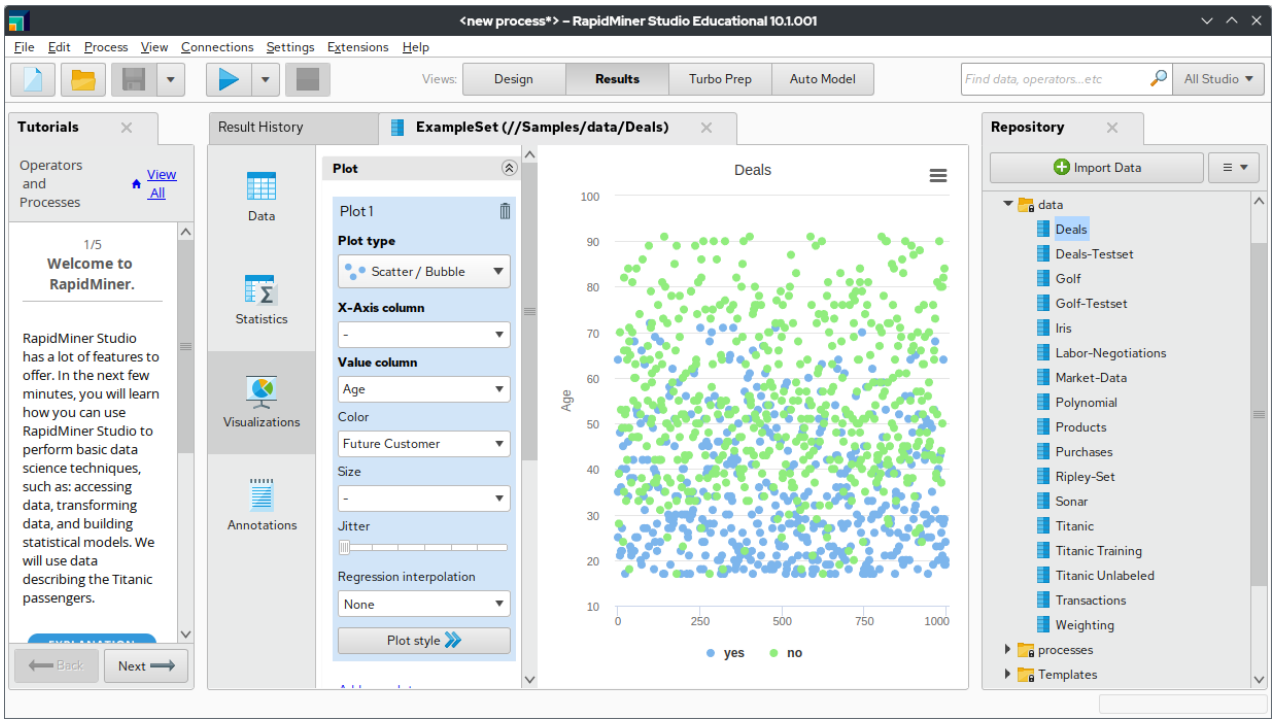


Рисунок 1.3 – Приклад графічного інтерфейсу RapidMiner з офіційного сайту

1.2.4 Порівняння існуючих рішень

Порівняння ПЗ для обробки великих наборів даних у вигляді таблиці наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння методів для пошуку спільнот користувачів у соціальних мережах

	Tableau	Microsoft Power BI	RapidMiner
Основне призначення	Візуалізація та аналітика даних	Бізнес-звітність та аналітика	Наука про дані, машинне навчання та аналітика
Основні недоліки	Висока вартість ліцензування; обмежений обсяг розширеної аналітики; зниження продуктивності при роботі з дуже великими наборами даних в режимі реального часу; для оптимізації потрібні досвідчені користувачі; немає вбудованого машинного навчання	Продуктивність обмежена через обмеження набору даних; складна крива навчання DAX; DirectQuery може працювати повільно; обмежена при використанні поза Microsoft stack; обмеження пам'яті в настільній версії	Високе споживання ресурсів; вимагає потужних машин; висока швидкість навчання; обмежена візуалізація; дорога корпоративна версія; не підходить для простих інформаційних панелей
Обробка великих обсягів даних	Середньо-висока (залежить від ресурсів сервера та вилучення даних)	Середньо-висока (добре працює зі службами Azure; слабка автономність)	Висока продуктивність (підтримує середовища Hadoop / Spark)
Можливості інтеграції	Багато з'єднувачів (Hadoop, Spark, SQL, хмарні сховища)	Чудово працює з продуктами Microsoft; багато зовнішніх з'єднань	Інтегрується з Hadoop, Spark, хмарними сховищами даних, базами даних SQL
Масштабованість	Масштабується за допомогою сервера Tableau / Cloud, але дорого	Добре масштабується тільки в рамках екосистеми Microsoft	Добре масштабується в корпоративних версіях і гірше в безкоштовній / настільній версії
Ідеально підходить для	Підприємства віддають перевагу візуалізації даних та інформаційним панелям BI	Організації, що використовують Microsoft stack та економічні робочі процеси BI	Спеціалістам з обробки даних потрібні наскрізні робочі процеси ML

Tableau, Microsoft Power BI і RapidMiner мають помітні недоліки в застосуванні до обробки великих обсягів даних. Tableau забезпечує чудову візуалізацію, але відчуває труднощі при роботі з надзвичайно великими наборами даних, особливо при використанні підключень у режимі реального часу замість отримання даних, і стає повільним без потужної серверної інфраструктури. У ньому також відсутні передові засоби машинного навчання, і його масштабування в масштабах всієї організації коштує дорого. Power BI є більш економічним, але має значні технічні обмеження: його настільна версія обмежена локальною пам'яттю, DirectQuery часто погано справляється зі складними запитами, а справжня масштабованість великих даних досяжна лише при глибокій інтеграції з хмарною екосистемою Microsoft, такою як Azure, Synapse або Fabric. Крім того, мова DAX від Power BI вимагає складного навчання, а гнучкість його візуалізації більш обмежена, ніж у Tableau.

RapidMiner, навпаки, хороший в машинному навчанні, але має свої недоліки в роботі з великими даними. Він ресурсомісткий, вимагає високопродуктивного обладнання для великих моделей і ефективно масштабується лише в поєднанні із зовнішніми системами обробки великих даних, такими як Hadoop або Spark. Його інтерфейс може виявитися складним для великих аналітичних робочих процесів, а можливості візуалізації мінімальні, що робить його непридатним для створення звітів BI. Нарешті, корпоративна версія RapidMiner коштує дорого, а безкоштовна версія не має багатьох розширених функцій, необхідних для масштабної аналітики.

Отже актуальною є задача розробки нового застосунку для обробки великих обсягів даних в реальному часі.

1.3 Постановка задачі

Метою роботи є розробка та дослідження нових засобів обробки великих наборів даних.

Під час роботи буде розроблено новий засіб обробки великих наборів даних для. Розглянемо більш детально перелік вимог до нового рішення:

а) обробка великого обсягу даних:

1) повинна підтримувати обробку структурованих, напівструктурованих і неструктурованих даних.

2) повинна обробляти безперервні потоки даних, пакетне завантаження і джерела подій в режимі реального часу;

б) інтеграція з масштабованим сховищем:

1) повинен підключатися до розподілених систем зберігання даних (HDFS, S3, Azure Data Lake, GCS);

2) повинен динамічно масштабуватися зі збільшенням обсягів даних;

3) повинен підтримувати секціонування та індексацію даних для великих наборів даних;

в) механізм розподіленої обробки даних:

1) повинен підтримувати паралельне, розподілене виконання завдань на декількох вузлах;

2) повинен забезпечувати як пакетну, так і потокову обробку даних;

г) відмовостійкість і відновлення:

1) повинен автоматично виявляти збої вузлів і перепризначати завдання;

2) повинен гарантувати одноразову обробку подій;

д) гнучке перетворення даних та ETL:

1) повинен підтримувати операції фільтрації, об'єднання, агрегації та управління вікнами;

2) повинен підтримувати еволюцію схеми та робочі процеси очищення даних;

е) розширена аналітика та інтеграція з машинним навчанням:

1) повинні надавати API або модулі для машинного навчання, прогнозування та виявлення аномалій;

2) повинні підтримувати навчання розподілених моделей і їх обслуговування;

є) інформаційна панель і моніторинг в режимі реального часу:

1) необхідно надавати показники продуктивності конвеєра, затримки, пропускної здатності і помилок;

2) необхідно надавати користувачам можливість відстежувати потоки обробки в режимі реального часу;

ж) управління ресурсами та оптимізація:

1) повинен автоматично розподіляти обчислювальні ресурси в залежності від робочого навантаження;

2) повинен підтримувати горизонтальне масштабування в розподілених кластерах;

3) повинен включати плани виконання з урахуванням витрат і продуктивності.

Нові рішення покращать та прискорять обробку великих наборів даних.

1.4 Висновки за розділом 1

В ході виконання першого розділу було досліджено проблемну область обробки BD.

Для подальшої роботи було проаналізована найбільш популярні фреймворки та екосистеми для роботи з BD, а також досліджено найбільш популярні сервіси та програмні комплекси по обробці BD. Додатково було проаналізовано найпопулярніші програмні рішення. На основі їхнього порівняння було прийнято рішення про розробку нового більш універсального рішення.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

Python став однією з найбільш широко використовуваних технологій для роботи з BD, і його популярність продовжує зростати в міру збільшення обсягів даних, джерел даних і складності аналізу. Основними причинами, чому Python виділяється, є його простота, велика екосистема бібліотек для маніпулювання даними та машинного навчання, сильна інтеграція з розподіленими обчислювальними платформами та здатність служити зв'язком між різними компонентами у великих потоках даних [19]. Крім того, Python – одна з небагатьох мов, яка однаково підходить для швидкого створення прототипів, обробки в масштабах виробництва, наукових досліджень і хмарних робочих процесів. Щоб зрозуміти, чому Python може бути найкращим рішенням для BD порівняно з іншими мовами та інструментами, важливо враховувати багато аспектів: продуктивність розробників, зрілість екосистеми, доступність бібліотек, підтримка спільноти, гнучкість інтеграції, узгодженість машинного навчання, сумісність із розподіленими системами та її роль у сучасних системах обробки даних інженерні архітектури [18].

Однією з головних причин, чому Python добре працює з BD, є його простота використання. Ця мова проста, виразна та читабельна, що дозволяє командам реалізовувати складну логіку в меншій кількості рядків коду, ніж мови низького рівня, такі як Java або C++. Така простота знижує когнітивне навантаження на розробників і значно спрощує обслуговування великих кодових баз [18]. У контексті великих даних, де потоки можуть постійно розвиватися, така гнучкість необхідна. Кожен етап роботи з BD: прийом даних, очищення, перетворення, статистичний аналіз, вилучення функцій, машинне навчання та розгортання – часто вимагає коригувань та експериментів. Python робить цей процес менш болючим, особливо для команд, до складу яких входять фахівці з обробки даних, аналітики та інженери з різним рівнем досвіду програмування. Замість того, щоб змусити всіх вивчати мови системного рівня,

Python надає доступний спосіб швидкого написання потужної логіки обробки даних [18].

Ще однією ключовою перевагою Python є велика екосистема бібліотек для обробки даних. Такі інструменти, як NumPy, pandas, Polars та Dask, забезпечують ефективні способи роботи з BD, пропонуючи абстракції високого рівня для таких операцій, як фільтрація, групування, агрегування та обчислення матриць. Ці бібліотеки розробляються з урахуванням продуктивності, часто використовуючи оптимізовані ядра C або Rust [21]. Це означає, що розробники можуть писати простий синтаксис на Python, використовуючи при цьому низькорівневість. У порівнянні з ручними реалізаціями структур даних на мовах, що не містять таких бібліотек, Python забезпечує продуктивність і результативність в поєднанні. Що ще важливіше, багато з цих бібліотек призначені для масштабування за межі пам'яті. Наприклад, Dask може розбивати великі набори даних на блоки та обробляти їх паралельно на одній машині або в розподіленому кластері. Polars використовує високооптимізовані фрейми даних, які дозволяють обробляти надзвичайно великі джерела даних з дивовижною ефективністю. Ці можливості дозволяють Python обробляти як малі, так і масивні набори даних, не вимагаючи використання інших мов чи інструментів [20].

Python також глибоко інтегрований з розподіленими обчислювальними платформами, що робить його практичним вибором для масштабної обробки даних. Сучасні платформи обробки великих даних, такі як Apache Spark, Apache Flink, Apache Beam, Ray та Hadoop, пропонують API Python [20]. Це дозволяє розробникам створювати розподілені завдання, використовуючи знайомий синтаксис Python, і при цьому використовувати можливості великих кластерів. Наприклад, PySpark дозволяє конвертувати дані розміром у петабайт, використовуючи той самий концептуальний інтерфейс, що і pandas. Ray надає платформу розподілених обчислень, розроблену спеціально для робочих навантажень на Python, що робить доступним широкомасштабний паралелізм для машинного навчання та конвеєрів передачі даних [20]. Пакет Apache Beam

Python SDK забезпечує уніфіковані пакетні та потокові конвеєри, які можуть виконуватися на декількох серверних платформах, таких як Dataflow або Flink. Навіть Hadoop, спочатку створений для Java, тепер підтримує Python для робочих процесів MapReduce за допомогою потокової передачі даних Hadoop. Така сумісність з екосистемою дає Python унікальну перевагу: його можна використовувати як мову, що поєднує багато середовищ виконання та механізми обробки великих даних [20].

Ще однією важливою особливістю Python при роботі з BD є його домінуюче становище в машинному навчанні та статистичному моделюванні. Переважна більшість сучасних фреймворків штучного інтелекту вбудовані або створені для Python, включаючи TensorFlow, PyTorch, scikit-learn, XGBoost, LightGBM та багато інших [21]. Машинне навчання є важливим компонентом аналітики BD, оскільки для роботи з великими масивами даних часто потрібні прогностичні моделі, виявлення закономірностей, прогнозування та виявлення аномалій. Python забезпечує плавну інтеграцію між конвеєрами обробки даних та робочими процесами машинного навчання. Замість того, щоб витягувати дані з однієї системи, перетворювати їх в іншу та створювати моделі в третьому середовищі, команди можуть робити все в межах однієї екосистеми. Конвеєр великих даних на базі Python дозволяє завантажувати великі набори даних з розподіленого сховища, попередньо обробляти дані за допомогою pandas або Dask, навчати модель за допомогою TensorFlow або scikit-learn, а потім розгортати її за допомогою FastAPI або хмарної платформи. Такий наскрізний робочий процес зменшує тертя і усуває необхідність перемикання мов або інструментів, що прискорює розробку і зменшує кількість помилок [21]. Оскільки Python домінує в дослідженнях штучного інтелекту, нові алгоритми і методи майже завжди з'являються на ньому першими, що гарантує фахівцям по роботі з великими даними ранній доступ до новітніх методів.

Python також добре інтегрується з хмарними екосистемами, які стали основним середовищем для обробки BD. Всі великі хмарні провайдери-AWS, Google Cloud, Microsoft Azure — пропонують розширені пакети Python SDK, що

дозволяють розробникам програмно взаємодіяти з базами даних, рівнями зберігання, потоковими системами і обчислювальними ресурсами [19]. Сценарії Python дозволяють легко керувати складними робочими процесами обробки даних, які охоплюють хмарні сховища, безсерверні функції, кластери Kubernetes та керовані послуги великих даних, такі як BigQuery, Redshift, Synapse та Snowflake. Така гнучкість гарантує, що Python може служити не просто інструментом обробки, а й сполучною мовою, що об'єднує компоненти сучасних архітектур обробки даних. Сумісність Python з інструментами інфраструктура як код, такими як Terraform і Pulumi, ще більше підсилює цю можливість, дозволяючи командам програмно налаштовувати і розгортати середовища обробки BD [20].

Більше того, розвиток спільноти та екосистеми Python значною мірою сприяє його перевагам. Величезна глобальна спільнота постійно розробляє бібліотеки, документацію, навчальні посібники та практичні рішення, які допомагають як новачкам, так і експертам [22]. Вирішуючи проблеми, пов'язані з великими обсягами даних, такими як розподілена обробка, вузькі місця в пам'яті, потокове передавання або машинне навчання, розробники можуть знайти безліч інструментів із відкритим кодом та прикладів із спільноти. Наявність такої великої екосистеми скорочує час, необхідний для вирішення проблем, і покращує довгострокову підтримку. Навпаки, багатьом альтернативним мовам або інструментам обробки великих даних не вистачає такої широкої підтримки, що робить розробку повільнішою та менш гнучкою [21].

Гнучкість – ще один вагомий аргумент для Python як рішення для великих даних. Хоча деякі інструменти, такі як SQL, чудово підходять для запитів та агрегування даних, вони не ідеальні для складних алгоритмів, динамічної логіки або даних, які не вписуються в реляційні моделі. І навпаки, такі мови, як Java або Scala, є потужними, але вимагають більше шаблонного коду і вимагають більш складного навчання [19]. Python зручно розташовується між цими крайнощами, здатний виражати як просту, так і дуже складну логіку.

Така гнучкість робить Python корисним у гетерогенних середовищах обробки даних, де набори даних представлені у безлічі форм, а завдання обробки можуть вимагати користувацьких перетворень. Python також підтримує багато парадигм програмування, включаючи об'єктно-орієнтований, функціональний та процедурний стилі, що дозволяє йому адаптуватися до широкого спектру шаблонів дизайну для обробки даних [19].

Крім того, Python все частіше використовується для обробки даних у режимі реального часу. Такі бібліотеки, як Faust, Streamz та спеціальні консолі Python для Kafka або Pulsar, дозволяють Python працювати з потоковими даними в архітектурах, керованих подіями. Хоча історично в потоковому потоці домінували фреймворки на основі Java, останні досягнення в потоковій обробці на основі Python та її інтеграції з розподіленими двигунами, такими як Spark Structured Streaming або API Flink Python, зробили Python прийнятним рішенням для робочих навантажень з низькою затримкою [19]. Хоча Python не завжди є найшвидшою мовою простого виконання, його здатність делегувати обчислення оптимізованим нативним бібліотекам або розподіленим системам компенсує це обмеження. Поєднання простоти та масштабованості робить Python надзвичайно конкурентоспроможним навіть в умовах, чутливих до затримок [22].

Нарешті, Python сприяє швидкому прототипуванню та експериментуванню, що є надзвичайно цінним у великих проєктах даних. Багато ініціатив, пов'язаних з великими даними, вимагають вивчення нових джерел даних, тестування декількох стратегій попередньої обробки або оцінки різних моделей машинного навчання. Python дозволяє командам швидко виконувати ітерації, локально запускаючи Прототипи на невеликих зразках, а потім масштабуючи їх у розподілених середовищах з мінімальними змінами коду. Ця можливість починати з малого і розширювати масштаб без перемикання інструментів або переписування логіки є основною перевагою для підвищення продуктивності та інновацій [19].

Таким чином, Python може стати найкращим рішенням для великих даних, оскільки він поєднує простоту, гнучкість, потужну екосистему, сумісність із розподіленими системами та глибоку інтеграцію з машинним навчанням. Він дозволяє розробникам писати зрозумілий, зручний в обслуговуванні код, який ефективно виконується завдяки оптимізованим вбудованим бібліотекам і серверним системам розподілених обчислень. Python усуває розрив між розробкою даних, наукою про дані, машинним навчанням та хмарними обчисленнями, що робить його ідеальною мовою для сучасних процесів обробки великих даних [20]. Це зменшує фрагментацію між інструментами, прискорює розробку та дозволяє командам створювати масштабовані інтелектуальні системи, керовані даними, з меншою кількістю бар'єрів. Оскільки обсяг, різноманітність і швидкість обробки даних продовжують зростати, роль Python як об'єднуючої технології стає ще більш цінною, гарантуючи, що вона залишиться в центрі обробки великих даних на довгі роки вперед [20].

Обґрунтування вибору мови програмування для розробки та дослідження нових засобів обробки великих наборів даних наведено таблиці 2.1.

Python, Scala та Java відіграють важливу, але різну роль у розробці інструментів обробки великих даних, і їх сильні сторони залежать від різних типів робочих навантажень, командних навичок та системних вимог. Python відрізняється простотою, швидкою швидкістю розробки та надзвичайно багатою екосистемою для аналізу даних та машинного навчання. Це найдоступніша мова для змішаних команд інженерів, аналітиків та науковців даних. Певні недоліки Python пов’язані з обмеженнями швидкості виконання в оригінальному вигляді та залежність від зовнішніх фреймворків, таких як PySpark або Dask, для ефективного масштабування. Але саме ця наявність великої кількості сторонніх бібліотек та фреймворків значно розширює можливості Python та спрощує розробку на цій мові [20].

Scala пропонує зовсім інші переваги: високу продуктивність на JVM, вбудовану підтримку Apache Spark і потужні можливості паралелізму завдяки функціональному програмуванню і платформі Akka framework. Вона високоефективна для потокових і розподілених систем і широко використовується у внутрішніх компонентах движків обробки великих даних. Однак найбільшими проблемами Scala є складність вивчення, більш складний синтаксис та значно менша спільнота порівняно з Python або Java. З цих причин Scala ідеально підходить для створення вузькоспеціалізованих високопродуктивних розподілених рушіїв або застосунків, але менш підходить для груп обробки даних, які потребують швидкої ітерації [22].

Java залишається значним конкурентом в інфраструктурі великих даних, оскільки багато основних платформ, включаючи Hadoop, Flink та Kafka, написані на Java. Java забезпечує високу продуктивність, відмінну оптимізацію JVM і стабільні можливості паралелізму корпоративного рівня. Недоліки Java пов’язані зі складністю: деталізований код, низька швидкість розробки та обмежені бібліотеки машинного навчання, що робить цей вибір менш придатним для дослідницької аналітики. Java найкраще використовувати при створенні надійних низькорівневих масштабних корпоративних систем, що вимагають передбачуваності, зручності обслуговування та довгострокової стабільності [21].

Таким чином, незважаючи на певні незручності – Python залишається найкращим вибором для побудови інтелектуальних конвеєрів даних, створення прототипів робочих процесів з великими даними та інтеграції машинного навчання в системи обробки великих даних (табл. 2.1).

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки методів реконструкції тривимірних зображень

	Python	Scala	Java
Основні переваги	Простота написання; багата екосистема машинного навчання; гнучкість; швидке створення прототипів	Рідна мова для Spark; висока продуктивність; функціональність; компілюється	Стабільність; висока продуктивність
Рівень продуктивності	Середнє (інтерпретується, більш повільне виконання в початковому вигляді)	Швидко (працює на JVM, оптимізовано для Spark)	Від швидкого до надшвидкого (оптимізовано для JVM, потужний компілятор JIT)
Поріг входження в розробку (складність вивчення)	Низький (легко вивчається)	Високий (складна у вивченні)	Середній (може бути адаптовано під попередній досвід)
Екосистема для великих даних	Багато бібліотек та додаткових фреймворків	Середня кількість бібліотек та додаткових фреймворків	Середня кількість бібліотек та додаткових фреймворків
Найкращі варіанти використання	Обробка даних, конвеєри на основі ML, оркестровка, прототипування	Інструменти потокової передачі в реальному часі, розподілені двигуни, рідні програми Spark	Пакетні двигуни, розподілені системи низького рівня, корпоративні платформи обробки великих даних
Типова швидкість розвитку	Дуже швидкий (висока продуктивність)	Помірна (більш складна в розробці)	Повільніше (більше шаблонів)
Ефективність використання пам'яті	Висока	Висока	Висока
Підтримка спільноти	Потужна спільнота	Спільнота розвинена, але не активна	Спільнота розвинена, але не активна

Отже, для реалізації ПЗ для обробки великих наборів даних обрано мову програмування Python.

2.2 Використання зовнішніх бібліотек та фреймворків

PySpark є однією з найбільш широко використовуваних технологій для розробки програмних рішень для обробки BD, оскільки вона забезпечує

інтерфейс Python для Apache Spark, розподіленого обчислювального механізму, призначеного для великомасштабної обробки даних [22]. PySpark дозволяє розробникам писати код на Python, який працює на кластерах комп'ютерів, дозволяючи програмам ефективно обробляти терабайти або петабайти даних. Основна причина, чому PySpark настільки популярний у сучасній обробці даних, полягає в тому, що він поєднує простоту Python з масштабованістю Spark. Це означає, що команди можуть створювати складні та високопаралельні конвеєри обробки даних, використовуючи знайомий синтаксис Python, і при цьому отримувати вигоду від здатності Spark розподіляти завдання на багатьох вузлах [22].

Щоб зрозуміти, як PySpark використовується для розробки програмного забезпечення для великих даних, корисно почати з основ архітектури Spark. Spark використовує гнучку модель розподіленого набору даних та механізм розподіленого виконання, який розбиває операції з даними на завдання, планує їх у кластері та автоматично обробляє збої [22]. PySpark не виконує код Python безпосередньо на розподілених вузлах; натомість він перетворює інструкції Python в операції Spark. Ці операції виконуються в ядрі Spark на основі JVM, тоді як користувацькі функції в Python серіалізуються та виконуються там, де це необхідно, за допомогою робочих процесів. Як результат, розробники можуть писати код, який виглядає як звичайна обробка даних на Python, але насправді виконується як розподілені паралельні обчислення [22].

Розробка застосунків для обробки BD за допомогою PySpark зазвичай починається зі створення SparkSession, який є з'єднанням із кластером Spark. Цей сеанс використовується для читання даних із розподілених систем зберігання даних, таких як HDFS, Amazon S3, Azure Data Lake, Google Cloud Storage або розподілених файлових систем у локальних кластерах. PySpark дозволяє розробникам читати безліч форматів, включаючи CSV, JSON, Parquet, ORC, Avro та бази даних через JDBC [22]. Після завантаження даних ядро обробки PySpark будується навколо фреймворків даних, які є розподіленими таблицями, оптимізованими для паралельних операцій. DataFrames надає API

високого рівня для фільтрації, об'єднання, агрегування, групування, очищення та перетворення великих наборів даних за допомогою синтаксису, подібного до SQL. Оскільки Spark оптимізує ці операції, розробники можуть писати компактний код на Python, а Spark визначає найбільш ефективний план виконання [23].

Потужність PySpark стає особливо помітною при роботі з даними, обсяг яких перевищує обсяг пам'яті однієї машини. Традиційні бібліотеки Python, такі як pandas, обмежені локальною пам'яттю, але фрейми даних PySpark розподілені по кластеру, що дозволяє набору даних рости практично нескінченно. Spark автоматично розділяє дані та планує завдання, а це означає, що розробникам не потрібно вручну писати логіку розпаралелювання. Виконуючи такі операції, як зіставлення, зменшення, об'єднання або групування, Spark обробляє розділення та переміщення даних у масштабі. Розробники PySpark зосереджені на описі перетворень, а не на тому, як розподілити роботу [23].

PySpark також підтримує SQL через модуль Spark SQL. Це дозволяє розробникам запускати SQL-запити безпосередньо в розподілених файлах даних або реєструвати їх як тимчасові таблиці для більш складної обробки. Запити SQL оптимізуються за допомогою оптимізатора Catalyst Spark, який перетворює запити в ефективні розподілені плани виконання. Це робить PySpark корисним як для інженерів даних, які віддають перевагу Python, так і для аналітиків, які віддають перевагу SQL, забезпечуючи можливість спільної розробки. Інтеграція SQL також дозволяє командам переводити застарілі системи на базі SQL у масштабовані конвеєри без Переписування цілих логічних структур [23].

Крім пакетної обробки, PySpark забезпечує потужну підтримку потокової передачі даних за допомогою структурованої потокової передачі. Ця система розглядає потоки даних у режимі реального часу як постійно оновлюванні таблиці та дозволяє застосовувати ті самі операції з фреймами даних до вхідних даних [23]. Потокові програми PySpark використовуються для

аналітики в режимі реального часу, наприклад, для виявлення шахрайства, моніторингу подій, обробки даних з датчиків IoT та інтерактивних інформаційних панелей. Використовуючи структуровану потокову передачу, розробники пишуть код один раз, а Spark автоматично обробляє мікропотоки, управління станом, контрольні точки та відновлення. Це значно полегшує розробку застосунків для обробки великих даних у режимі реального часу [23].

Ще однією важливою складовою PySpark в розробці програмного забезпечення для роботи з великими даними є можливості машинного навчання за допомогою бібліотеки MLlib. MLlib включає масштабовані алгоритми для класифікації, регресії, кластеризації, рекомендацій та розробки функцій [25]. Ці алгоритми реалізовані в розподіленій архітектурі Spark, що означає, що великі завдання з навчання моделей можуть виконуватися в кластерах. Розробники можуть попередньо обробляти дані за допомогою фреймворків даних PySpark, витягувати функції за допомогою вбудованих перетворювачів та навчати моделі за допомогою оцінювачів MLlib. Однак PySpark MLlib не замінює бібліотеки Python ML, такі як scikit-learn або TensorFlow; натомість він забезпечує навчання розподіленої моделі, коли набори даних занадто великі для однієї машини. Розробники часто використовують PySpark для масової підготовки даних, а потім експортують оброблені набори даних у фреймворки ML, призначені для глибокого навчання або спеціалізованих алгоритмів [26].

PySpark також часто використовується для побудови потоків ETL у галузі розробки великих даних. Робочі процеси ETL включають вилучення даних з декількох джерел, їх перетворення для очищення та збагачення вмісту та завантаження в сховища або аналітичні системи. PySpark автоматизує більшу частину цього процесу, дозволяючи виконувати паралельні операції читання, запису та перетворення. Потоки ETL, побудовані за допомогою PySpark, зазвичай організовуються за допомогою таких інструментів, як Airflow, Luigi або хмарні оркестратори [25]. Ці конвеєри можуть бути заплановані для періодичного запуску або запускатися при надходженні нових даних. Оскільки завдання Spark можуть бути упаковані як автономні програми, код PySpark

може бути розгорнутий у кластерах, середовищах Kubernetes або хмарних платформах, таких як AWS EMR, Databricks, Google Dataproc або Azure Synapse.

На додаток до традиційного ETL, PySpark широко використовується для створення рівнів обробки data lake. Data lake зберігає необроблені дані у файлових системах, таких як S3 або ADLS, а PySpark дозволяє конвертувати ці необроблені дані у ретельно відібрані структуровані шари. Це часто реалізується за допомогою архітектури medallion, де бронзові таблиці зберігають необроблені дані, срібні таблиці очищають дані, а золоті таблиці зберігають зведені дані, готові для бізнесу. Здатність PySpark обробляти великі таблиці Parquet або Delta Lake робить його ідеальним для створення масштабованих перетворень даних, які розвиваються з часом [24].

Інтеграція PySpark з хмарними середовищами – ще одна причина, чому він використовується для розробки програмного забезпечення для обробки великих даних. Керовані сервіси Spark знижують операційні витрати, пов'язані з запуском кластерів, і дозволяють розробникам зосередитися на логіці програми. На хмарних платформах сценарії PySpark можна запускати в інтерактивному режимі в блокнотах, пакетних завданнях або повністю автоматизованих конвеєрах. Екосистема Python доповнює це, надаючи інструменти для інтеграції API, машинного навчання, оркестровки та візуалізації [24].

Одним з важливих факторів при розробці за допомогою PySpark є налаштування продуктивності. Хоча код PySpark може нагадувати код Python, продуктивність залежить від розподіленої оптимізації Spark. Розробники повинні розуміти такі поняття, як розділення, перемішування, кешування, трансляція та обмеження пам'яті, щоб забезпечити ефективне виконання. Наприклад, неправильні операції об'єднання можуть призвести до дорогого перемішування даних. І навпаки, використання широкомовних об'єднань або кешування часто використовуваних фреймів даних може значно підвищити швидкість. Для досягнення оптимальних результатів PySpark вимагає балансу між логікою на рівні Python та розподіленим плануванням на рівні Spark [25].

Таким чином, PySpark використовується для розробки програмного забезпечення для обробки великих обсягів даних, забезпечуючи інтерфейс на основі Python для двигуна розподілених обчислень Spark. Він забезпечує масштабовану пакетну та потокову обробку даних, інтегрується з SQL, підтримує розподілене машинне навчання та безперешкодно вписується в сучасні архітектури ETL та data lake. Поєднання простоти використання, надійних можливостей та розподіленої масштабованості робить його центральною технологією в сучасній екосистемі великих даних. PySpark дозволяє розробникам ефективно обробляти величезні масиви даних без складного паралельного програмування, дозволяючи командам створювати надійні, масштабовані та високопродуктивні системи обробки даних, використовуючи знайомі інструменти Python [26].

2.3 Вибір середовища для розробки

PyCharm широко розглядається як одне з найефективніших інтегрованих середовищ розробки (integrated development environment – IDE) для Python, особливо при роботі з бібліотеками та фреймворками, що вимагають великих обсягів даних, таких як Pandas, PySpark та MLlib. Його перевага полягає в поєднанні вдосконаленого аналізу коду, надійних інструментів структурування проєктів, безперебійного управління середовищем та потужної інтеграції з робочими процесами, що працюють з великими даними [27]. Хоча багато розробників також використовують більш легкі інструменти, такі як VS Code, Jupyter Notebook або Cloud notebook, PyCharm надає більш комплексне та орієнтоване на виробництво середовище, яке підтримує весь життєвий цикл розробки програмного забезпечення для роботи з великими даними – від створення прототипів до створення великих розподілених застосунків [27].

Однією з найбільших переваг PyCharm є інтелектуальна підтримка коду. IDE забезпечує глибоку перевірку коду та статичний аналіз, пропонуючи високоточне завершення, підказки щодо набору тексту та інструменти

рефакторингу, які допомагають розробникам писати чистіший та надійніший код. Це особливо корисно при написанні складних конвеєрів перетворення даних за допомогою Pandas або PySpark [27], де невеликі помилки в іменах стовпців, типах або операціях можуть порушити весь робочий процес. PyCharm аналізує імпорт, структури об'єктів і підписи методів, що спрощує навігацію по коду і дозволяє уникнути помилок на ранній стадії. Це особливо важливо в PySpark, де API обширний і іноді менш задокументований, ніж в основних бібліотеках Python, і де помилки зазвичай з'являються під час виконання, а не під час компіляції. Перевірки PyCharm допомагають виявити потенційні проблеми в операціях з фреймворками даних, користувацьких функціях та конвеєрах MLlib перед їх виконанням у кластері, заощаджуючи час та обчислювальні ресурси .

Управління середовищем та залежностями – ще одна сильна сторона PyCharm. Проекти з великими даними часто вимагають різних інтерпретаторів Python, віртуальних середовищ або навіть підключення до віддалених кластерів. PyCharm підтримує virtualenv, Conda, Pipenv, інтерпретатори на основі Docker та віддалені інтерпретатори на основі SSH. Для локальної обробки даних на основі Pandas розробник може використовувати простий virtualenv з мінімальними залежностями [27]. Для розробки PySpark проекту може знадобитися підключення до зовнішньої інсталяції Spark або віддаленого сервера. PyCharm може легко налаштувати ці середовища, забезпечуючи послідовне виконання сценаріїв у локальних та розподілених установках. Він також може виявляти конфлікти пакетів та відсутні залежності, що є надзвичайно важливим для проектів даних, які залежать від конкретних версій бібліотек у великих наборах даних та конвеєрах [27].

Під час роботи з PySpark та MLlib налагодження може бути особливо складним, оскільки завдання Spark часто виконуються в розподілених кластерах, а помилки в перетвореннях або діях можуть з'являтися лише при віддаленому виконанні завдання. PyCharm надає вдосконалені засоби налагодження, які дозволяють використовувати точки зупинки, крок за кроком,

перевірку змінних та оцінку середовища виконання. Це стає надзвичайно цінним при розробці додатків Spark локально перед їх розгортанням на повних кластерах. Розробники можуть тестувати перетворення, перевіряти етапи навчання моделі або вивчати проблеми зі схемою в PySpark DataFrames – все в IDE. PyCharm також підтримує налагодження для віддалених інтерпретаторів Python, що означає, що розробники можуть налагоджувати завдання, що виконуються в кластерному середовищі, безпосередньо зі свого локального комп'ютера – потужна функція для конвеєрів обробки великих даних корпоративного рівня [27].

Інтегроване середовище розробки також забезпечує потужну підтримку роботи зі структурованими даними. Вбудовані засоби роботи з базами даних дозволяють розробникам підключатися до баз даних SQL, сховищ даних і зовнішніх джерел даних, переглядати таблиці, виконувати запити і візуалізувати дані безпосередньо в середовищі IDE. Це доповнює робочі процеси Pandas і PySpark: фахівці з обробки даних можуть вивчати необроблені дані, виконувати запити попередньої обробки SQL і негайно завантажувати результати в Python для подальшого аналізу. PyCharm також надає інтерактивні інструменти, такі як науковий режим та переглядачі даних, які дозволяють швидко візуалізувати фрейми даних. Хоча блокнот Jupyter, як правило, популярніший для пошукового аналізу, PyCharm пропонує більш інтегрований та орієнтований на виробництво спосіб вивчення даних без перемикання інструментів [28].

Можливості PyCharm щодо організації проєктів особливо корисні для великомасштабних застосунків, що працюють з великими даними. Конвеєри обробки великих даних часто включають кілька модулів: сценарії ETL, модулі розробки функцій, навчальні конвеєри машинного навчання, сценарії оцінки та інструменти розгортання. Вони можуть швидко ускладнитися, якщо їх розподілити по декількох каталогах, скриптах і файлах конфігурації. PyCharm допомагає підтримувати чітку структуру за допомогою шаблонів проектів [28], настроюваної навігації по файлах, згортання коду та інструментів

рефакторингу. При роботі з кодовими базами PySpark, які включають десятки UDF-файлів, допоміжних класів і файлів конфігурації для кластерів, така організація скорочує кількість помилок і підвищує зручність обслуговування в довгостроковій перспективі [28].

Інтеграція з системами контролю версій, такими як Git, є ще однією важливою перевагою. У проєктах з BD часто беруть участь кілька розробників, які працюють на різних етапах розробки, і контроль версій стає необхідним. Вбудовані інструменти Git PyCharm дозволяють керувати розгалуженнями, вирішувати конфлікти, переглядати історію змін та порівнювати код без використання зовнішньої програми. Це покращує взаємодію та полегшує відстеження складних змін, що має вирішальне значення для середовищ, де часто відбуваються перетворення даних та моделі MLlib [28].

Що стосується машинного навчання за допомогою MLlib, PyCharm забезпечує потужне середовище для управління експериментами, відстеження версій моделей та інтеграції із зовнішніми платформами машинного навчання, такими як TensorFlow, scikit-learn або MLflow. Хоча MLlib спочатку забезпечує розподілені алгоритми в Spark, можливості PyCharm дозволяють розробникам інтегрувати Протоки MLlib з додатковою логікою обробки, попередньої обробки або оцінки на основі Python. Цей гібридний підхід-поєднання розподілених операцій MLlib з локальними інструментами Python ML – стає більш керованим завдяки послідовному середовищу розробки PyCharm [28].

Незважаючи на ці переваги, деякі розробники віддають перевагу легшим редакторам або середовищам на базі нотатників. Такі інструменти, як Jupyter, чудово підходять для швидкого експериментування, а VS Code пропонує гнучкість із розширеннями. Однак, створюючи масштабовані та підтримувані інструменти обробки великих обсягів даних, структуроване управління проектами PyCharm, глибокий аналіз Python та розширена налагодження дають йому значну перевагу. Це усуває розрив між експериментальною обробкою даних та розробкою програмного забезпечення виробничого рівня, що робить

його добре придатним для довгострокових проектів великих даних на основі Pandas, PySpark та MLlib [28].

Обґрунтування вибору середовища розробки для розробки нового методу обробки великих наборів даних наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки

Критерій порівняння мов програмування	Visual Studio	PyCharm
Підтримки Python	Може бути адаптована за допомогою надбудов та плагінів	Спеціалізоване IDE під Python
Простота використання	Загалом зручне IDE, але не зручне для розробки масштабованих застосунків	Оптимізовано під розробку будь-яких застосунків на Python
Кросплатформність	Windows та macOS; Linux – обмежено	Доступно для Windows, macOS і Linux.
Адаптивність середовища	Ресурсоємність збільшується при збільшенні сторонніх модулів	Легший та оптимізованіший за Visual Studio з набором плагінів, але може трішки сповільнятися при роботі з великими проєктами
Навігація по коду	Зручна навігація, розширена: IntelliSense для Python	Повністю адаптовано під Python
Розгортання фреймворків	Підтримує тестування на Python, але для інших надбудов потрібні додаткові налаштування	Підтримка фреймворків тестування за замовченням (unittest, pytest та nose)
Розширення / плагіни	Великий вибір сторонніх бібліотек та модулів (але може сповільнити завантаження)	Велика кількість перевірених сторонніх бібліотек та модулів
Управління віртуальним середовищем	Реалізовані тільки базові механізми	Автоматизоване визначення та налаштування віртуальних середовищ
Підтримка наукових досліджень даних	Є інтеграція з Jupyter Notebooks, але загалом не розраховано на наукові дослідження	Налаштована потужна інтеграція з безліччю сторонніх надбудов для дослідження даних

Таким чином, PyCharm кращий за багато альтернативних IDE для розробки великих даних на основі Python, оскільки забезпечує повне інтегроване середовище, яке підтримує аналіз коду, налагодження, тестування, управління середовищем, інтеграцію баз даних та складну організацію проекту.

Ці функції допомагають розробникам створювати надійні, підтримувані та масштабовані інструменти для обробки великих обсягів даних, починаючи з ранніх етапів експериментів і закінчуючи повним впровадженням у виробництво.

3 ОПИС ПРОГРАМИ

3.1 Загальна схема роботи

PySpark широко використовується при розробці застосунків для обробки великих обсягів даних, оскільки він забезпечує інтерфейс Python для Apache Spark, дозволяючи розробникам писати масштабований розподілений код, тоді як Spark обробляє паралельне виконання в кластері. Стандартна схема компонентів PySpark наведена на рис. 3.1. У більшості систем PySpark стає основним двигуном для потоків ETL, великомасштабної аналітики та робочих процесів машинного навчання. Типовим способом використання є зчитування даних із розподіленого сховища, перетворення їх за допомогою API DataFrame Spark, застосування аналітичних операцій або операцій машинного навчання та запис результатів назад у сховище або наступні системи.

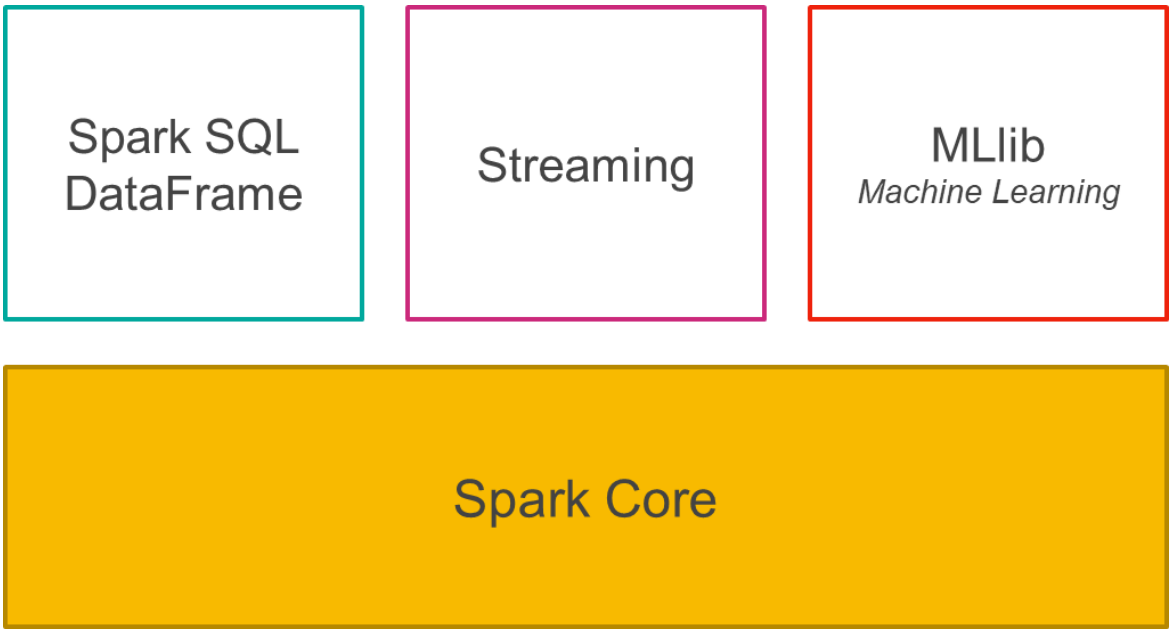


Рисунок 3.1 – Стандартні компоненти PySpark

Процес розробки зазвичай починається з прийому даних. Об'єкт SparkSession PySpark є основною точкою входу, забезпечуючи методи завантаження даних з таких джерел, як HDFS, Amazon S3, Azure big binary storage, потоки Kafka, бази даних SQL або файли parquet та CSV. Завантажені

дані представлені у вигляді фрейму даних Spark, який схожий на фрейм даних Pandas, але розподілений по багатьох вузлах. Потім розробники використовують високорівневі операції, такі як фільтрація, групування, об'єднання, агрегування та віконні функції, щоб перетворити необроблені вхідні дані в чисті, структуровані та збагачені дані, придатні для наступних етапів розробки.

Два найбільш часто використовувані модулі при розробці – PySpark SQL та PySpark Core. PySpark SQL надає API DataFrame та можливості запитів SQL, що дозволяє розробникам ефективно працювати зі структурованими даними. Він також включає функції для виразів стовпців, перетворення типів, операції з датою та часом та користувацькі функції. Ядро PySpark забезпечує низькорівневі операції з вибірками (Resilient Distributed Dataset – RDD), які дозволяють точно контролювати розподілені перетворення. Хоча фрейми даних частіше використовуються в сучасних розробках, оскільки вони оптимізовані за допомогою двигуна Spark Catalyst, RDD все ще відіграють важливу роль у завданнях, що вимагають розділення користувачів або маніпулювання на нижчому рівні.

Потокове передавання – ще одна важлива можливість. Структурована потокова передача PySpark дозволяє розробникам створювати конвеєри в режимі реального часу, які обробляють події з таких систем, як Kafka, Kinesis або socket sources. API використовує ті самі операції з фреймами даних, що і пакетна обробка, що означає, що розробники можуть повторно використовувати логіку для пакетних та потокових навантажень. Структурована потокова передача автоматично обробляє поступові обчислення, контрольні точки та стійкість до відмов, що робить її ідеальною для аналізу журналів, конвеєрів моніторингу та інформаційних панелей реального часу.

Машинне навчання підтримується за допомогою PySpark MLlib. Розробники використовують цей модуль для створення розподілених конвеєрів ML, які можуть обробляти набори даних набагато вище, ніж може обробляти

одна машина. MLlib надає алгоритми для класифікації, регресії, кластеризації, рекомендацій та розробки функцій. Він також включає інструменти для векторизації, нормалізації, обробки категоріальних змінних та конвеєрного ланцюга. Хоча для деяких завдань глибокого навчання можуть знадобитися спеціалізовані платформи, MLlib ефективний для класичного ML в масштабі.

PySpark також добре інтегрується з користувацькими функціями, Pandas UDF та векторизованими операціями. Це дозволяє розробникам застосовувати власну логіку Python для створення фреймворків даних Spark. UDF – файли Pandas особливо корисні, оскільки вони використовують Apache Arrow для ефективної передачі даних між Python та Spark, забезпечуючи високопродуктивні векторизовані операції. Ця можливість широко використовується для інтеграції бізнес-логіки або складних перетворень у розподілені конвеєри.

Іншою важливою частиною розробки на PySpark є оптимізація. Розробники часто використовують такі інструменти, як кешування, розділення, широкомовні об'єднання та об'єднання даних для підвищення продуктивності. PySpark SQL надає план виконання за допомогою функції explain (), що дозволяє розробникам виявляти вузькі місця та забезпечувати ефективну оптимізацію запитів у Spark. Параметри рівня кластера, такі як обсяг пам'яті виконавця, кількість ядер і конфігурації у випадковому порядку, також налаштовуються в залежності від характеристик робочого навантаження.

Нарешті, програми PySpark зазвичай записують свої результати назад у розподілене сховище, бази даних, черги повідомлень або інструменти подальшої аналітики. API запису підтримують багато форматів, таких як parquet, CSV, JSON, ORC, Delta Lake та JDBC. У багатьох виробничих процесах PySpark управляється менеджерами робочих процесів, такими як Airflow, Luigi або Databricks Jobs, для автоматизації навчання ETL, ML та потокових конвеєрів.

Таким чином, PySpark використовується протягом усього життєвого циклу розробки додатків для роботи з великими даними: прийом даних,

розподілене перетворення, потокове передавання, Машинне навчання, реалізація Користувацької логіки, оптимізація та зберігання. Основні модулі включають PySpark SQL для операцій з фреймворками даних, Pyspark Core для низькорівневих перетворень, PySpark MLlib для машинного навчання та Pyspark Structured Streaming для обробки в режимі реального часу. Таке поєднання робить PySpark одним з найпотужніших і гнучких інструментів для створення масштабованих систем обробки даних.

На схемі – рис. 3.2, показано розбудову застосунку на основі використання компонентів PySpark.

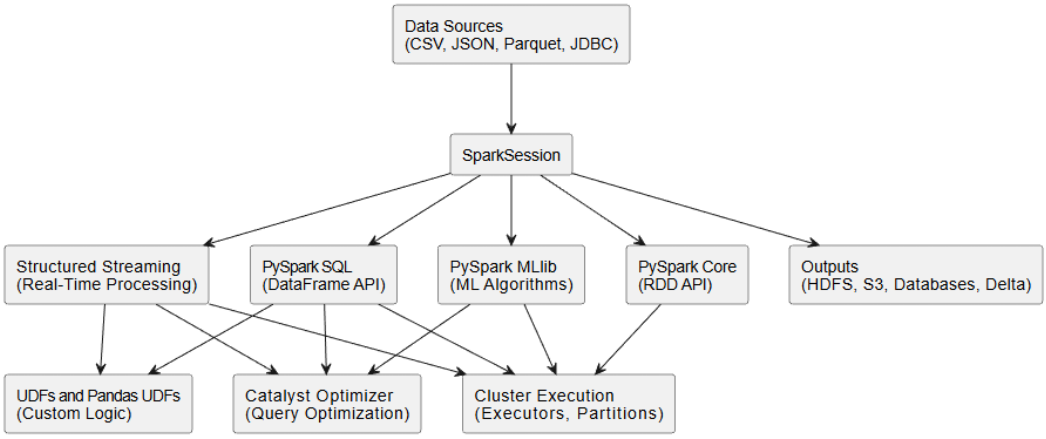


Рисунок 3.2 – Загальна схема нового рішення для обробки великих наборів даних, на основі PySpark

Діаграма класів на рис. 3.3 – це структурне представлення нового рішення для обробки великих наборів даних, на основі PySpark, що показує, як взаємодіють різні модулі (представлені як класи).

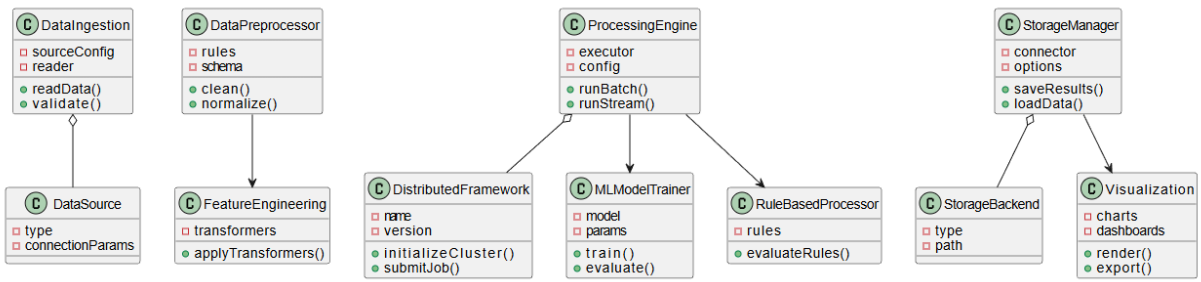


Рисунок 3.3 – Діаграма класів

3.2 Загальна взаємодія із системою

Загальний механізм взаємодії із системою наведено у вигляді логічного проєктування засобами UML – див. рис. 3.4.

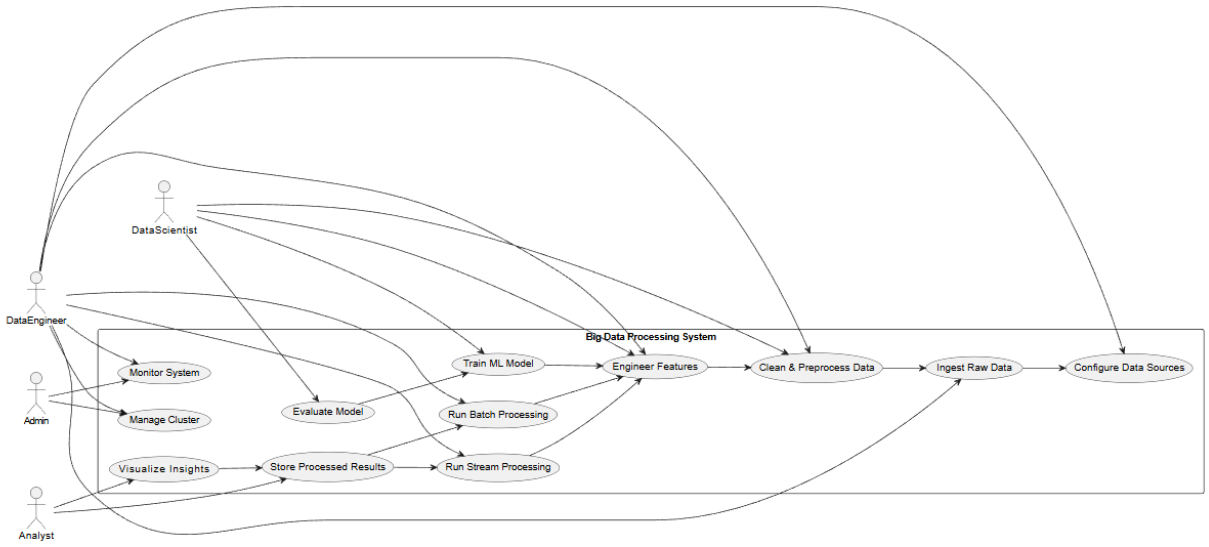


Рисунок 3.4 – Загальний механізм взаємодії із системою

Загалом, система моделює робочий процес синтезу діагностичних нейронних моделей. Вона підтримує попередню обробку даних, вибір функцій, синтез моделі, перевірку достовірності та розгортання. З системою взаємодіють як фахівці-люди, так і автоматизовані процеси. Експерт в предметній області – надає вимоги до предметної області, інтерпретує діагностичні цілі та контролює робочий процес моделювання. Інженер з обробки даних – готує та очищає набори даних, керує вибором функцій та забезпечує правильне форматування даних. Аналітик з діагностики – оцінює продуктивність моделі, перевіряє точність діагностики та затверджує моделі для розгортання.

Нейросинтезний движок – виконує еволюційний / оптимізаційний синтез нейронних архітектур. Виконує операції навчання, пошуку гіперпараметрів, кросинговеру та мутації.

Система моніторингу – відстежує продуктивність розгорнутої моделі, повідомляє про аномалії і запитує повторний синтез, коли це необхідно.

Основні варіанти використання системи:

– попередня обробка даних – включає очищення, нормалізацію, виявлення викидів, зменшення розмірності та керування версіями набору даних;

– вибір функцій – застосовує фільтр, оболонку, ансамбль або еволюційні стратегії перед синтезом;

– синтез нейронної моделі – генерує архітектури-кандидати, використовуючи методи нейросинтезу (наприклад, мутації, схрещування, оцінка придатності);

– оцінка та валідація моделі – перевіряє точність прогнозування, надійність, якість діагностики та продуктивність узагальнення;

– розгортання моделі – інтегрує перевірену модель в діагностичний конвеєр;

– перенавчання або повторний синтез моделі – опускається при зниженні продуктивності або зміні умов предметної області.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Метою роботи є розробка та дослідження нових засобів обробки великих наборів даних.

Під час роботи буде розроблено новий засіб обробки великих наборів даних для. Розоглянемо більш детально перелік вимог до нового рішення:

а) обробка великого обсягу даних:

1) повинна підтримувати обробку структурованих, напівструктурованих і неструктурованих даних;

2) повинна обробляти безперервні потоки даних, пакетне завантаження і джерела подій в режимі реального часу;

б) інтеграція з масштабованим сховищем:

1) повинен підключатися до розподілених систем зберігання даних (HDFS, S3, Azure Data Lake, GCS);

2) повинен динамічно масштабуватися зі збільшенням обсягів даних;

3) повинен підтримувати секціонування та індексацію даних для великих наборів даних;

в) механізм розподіленої обробки даних:

1) повинен підтримувати паралельне, розподілене виконання завдань на декількох вузлах;

2) повинен забезпечувати як пакетну, так і потокову обробку даних;

г) відмовостійкість і відновлення:

1) повинен автоматично виявляти збої вузлів і перепризначати завдання;

2) повинен гарантувати одноразову обробку подій;

д) гнучке перетворення даних та ETL:

1) повинен підтримувати операції фільтрації, об'єднання, агрегації та управління вікнами;

2) повинен підтримувати еволюцію схеми та робочі процеси очищення даних;

е) розширена аналітика та інтеграція з машинним навчанням:

1) повинні надавати API або модулі для машинного навчання, прогнозування та виявлення аномалій;

2) повинні підтримувати навчання розподілених моделей і їх обслуговування;

є) інформаційна панель і моніторинг в режимі реального часу:

1) необхідно надавати показники продуктивності конвеєра, затримки, пропускної здатності і помилок;

2) необхідно надавати користувачам можливість відстежувати потоки обробки в режимі реального часу;

ж) управління ресурсами та оптимізація:

1) повинен автоматично розподіляти обчислювальні ресурси в залежності від робочого навантаження;

2) повинен підтримувати горизонтальне масштабування в розподілених кластерах;

3) повинен включати плани виконання з урахуванням витрат і продуктивності.

Нові рішення покращать та прискорять обробку великих наборів даних.

4.2 Стратегія тестування

Для тестування було використано вибірку даних – FitRec_Dataset [29], [30]. Впорядкована версія для завдання прогнозування маршруту тренування складається з: 167373 тренування від 956 користувачів. Зразки тренувань включають часові позначки, різкі зміни координат GPS, результатів вимірювань, змінні: наприклад, швидкість і відстань.

Результати завантаження вибірки наведено на рис. 4.1.

	summary	gender	id	sport	url	userId
0	count	253020	253020	253020	253020	253020
1	mean	None	3.566244412926132E8	None	None	4619648.939783417
2	stddev	None	1.574845634895318E8	None	None	3932877.7296880507
3	min	female	99296	aerobics	https://www.endomondo.com/users/10014612/worko...	69
4	max	unknown	674008008	yoga	https://www.endomondo.com/users/9991401/workou...	15481421

There are total 253020 row, Let print first 2 data rows:

	altitude	gender	heart_rate	id	latitude	longitude	speed	sport	timestamp	url	userId
0	[41.6, 40.6, 40.6, 38.4, 37.0, 34.0, 34.0, 34....	male	[100, 111, 120, 119, 120, 116, 125, 128, 131, ...	396826535	[60.173348765820265, 60.173239801079035, 60.17...	[24.64977040886879, 24.65014273300767, 24.6509...	[6.8652, 16.4736, 19.1988, 20.4804, 31.3956, 3...	bike	[1408898746, 1408898754, 1408898765, 140889877...	https://www.endomondo.com/users/10921915/worko...	10921915
1	[38.4, 39.0, 39.0, 38.2, 36.8, 36.8, 36.8, 35....	male	[100, 105, 111, 110, 108, 115, 126, 130, 132, ...	392337038	[60.173247596248984, 60.17320962622762, 60.172...	[24.649855233728886, 24.65015547350049, 24.650...	[9.0792, 13.284, 15.9336, 10.9476, 16.1676, 30...	bike	[1408221682, 1408221687, 1408221699, 140822170...	https://www.endomondo.com/users/10921915/worko...	10921915

Рисунок 4.1 – Завантаження та відображення схеми вибірки даних

Огляд стовбців даних наведено на рис. 4.2.

	Column Name	Data type
0	altitude	array<double>
1	gender	string
2	heart_rate	array<bigint>
3	id	bigint
4	latitude	array<double>
5	longitude	array<double>
6	speed	array<double>
7	sport	string
8	timestamp	array<bigint>
9	url	string
10	userId	bigint

Рисунок 4.2 – Загальний огляд стовбців вибірки

Результати попереднього огляду вибірки з попередньою фільтрацією на рис. 4.3.

ОЗагальний підсумок набору даних про користувачів, тренування та кількість записів (попередня фільтрація):

	Users count	Activity types count	Workouts count	Total records count
0	1,104	49	253,020	111,541,956

Рисунок 4.3 – Загальний підсумок набору даних про користувачів, тренування та кількість записів (попередня фільтрація)

Відбір найбільш популярних видів спорту та графічне відображення на рис. 4.4 та 4.5.

Топ-5 видів спорту, в яких взяло участь найбільше користувачів:

	sport	Users count	percentage
0	run	865	25.130738
1	bike	794	23.067984
2	mountain bike	336	9.761766
3	bike (transport)	252	7.321325
4	walk	209	6.072051
5	others	986	28.646136

Рисунок 4.4 – Відбір найбільш популярних видів спорту

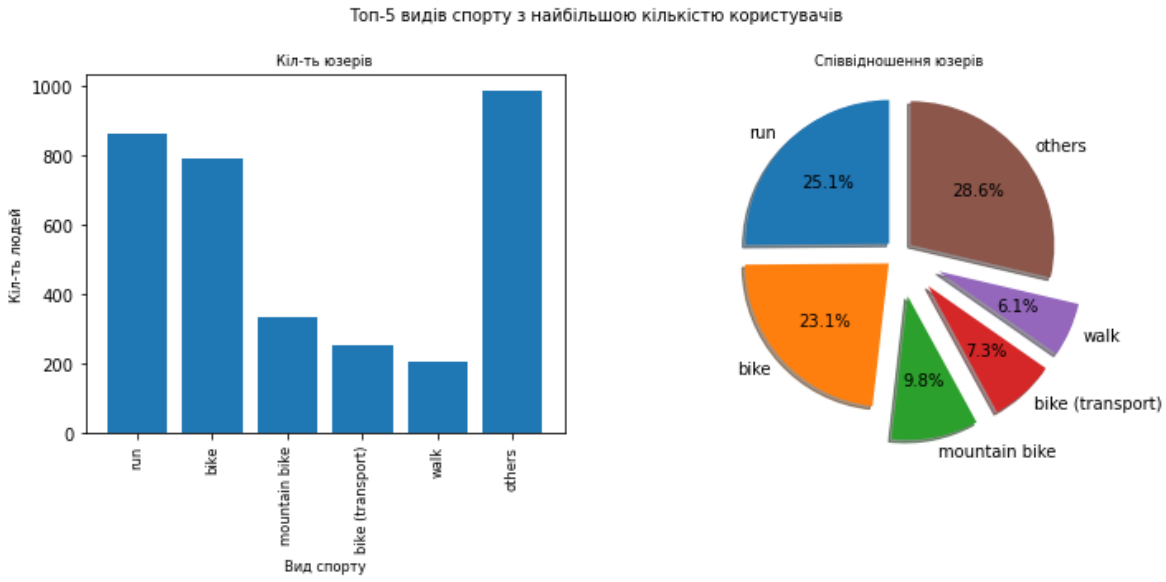


Рисунок 4.5 – Візуалізація інформації з вибірки

На рисунку 4.6 наведена діаграма розподілу за гендером.

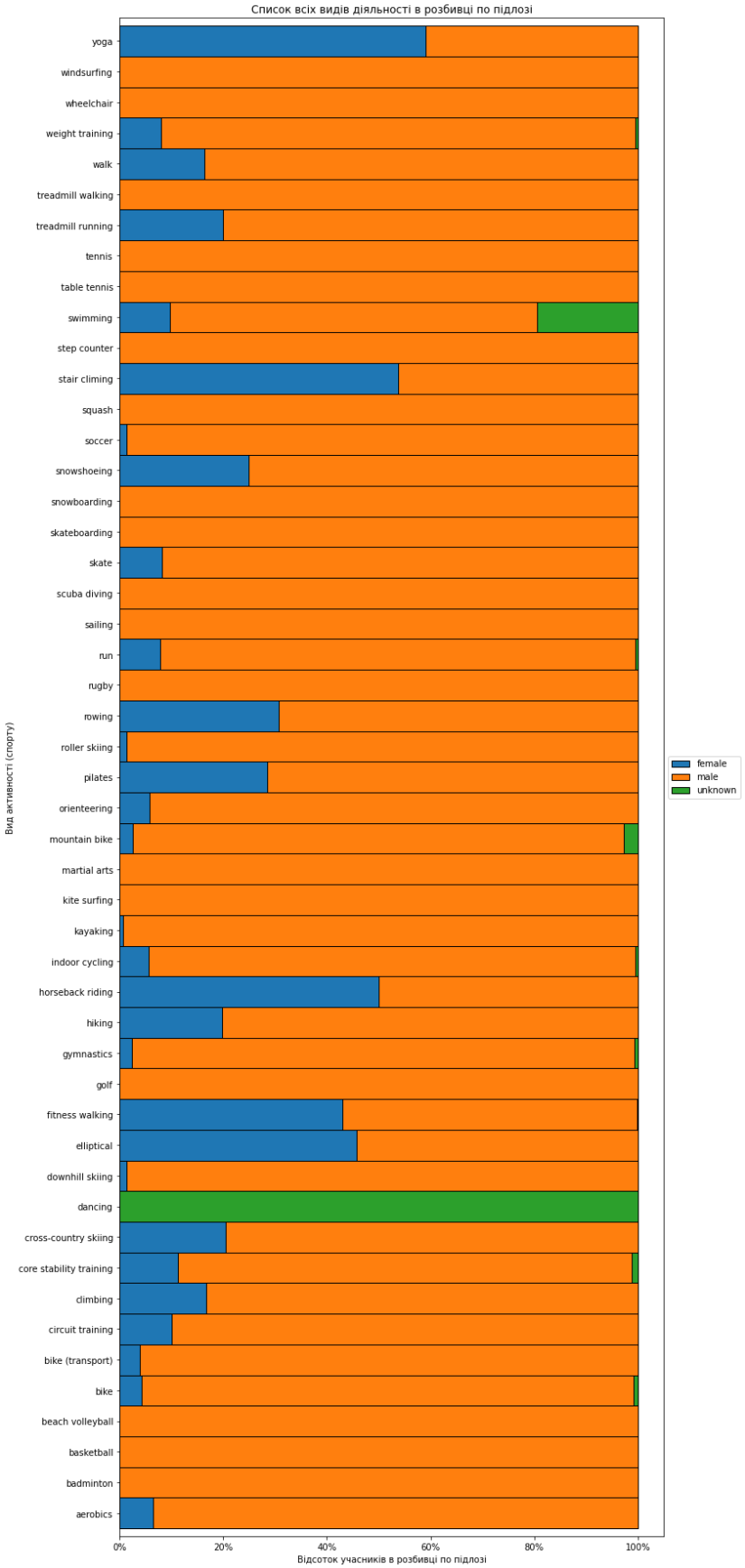


Рисунок 4.6 – Діаграма розподілу за гендером

На рис. 4.7 – 4.9 наведено часткове відображення розпорошення даних за значенням цільової ознаки.

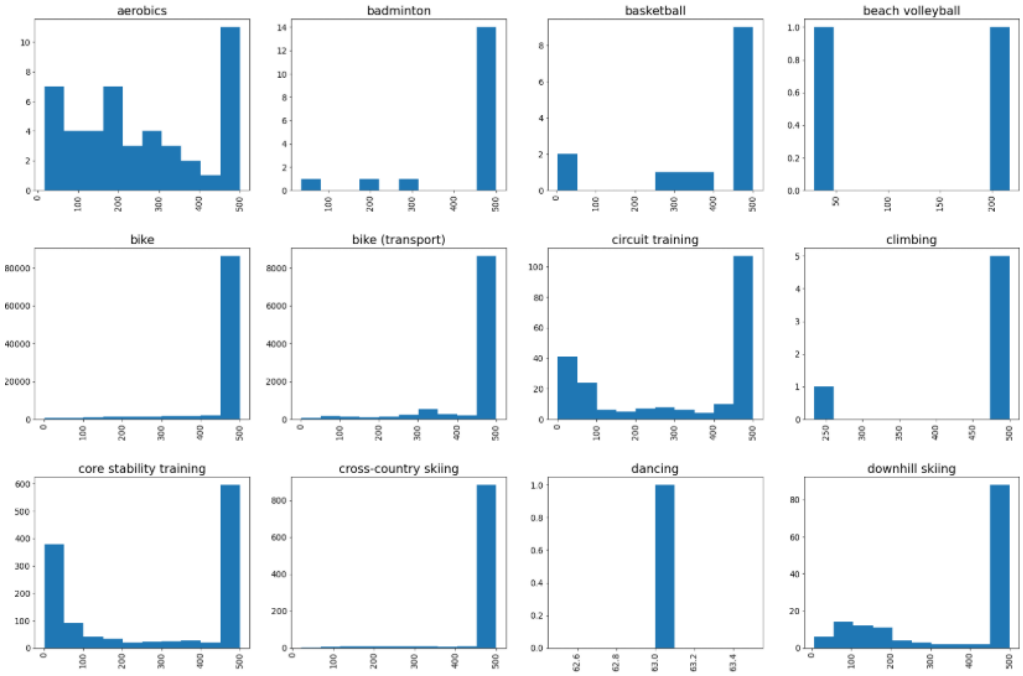


Рисунок 4.7 – Приклад відображення розпорошення даних за значенням цільової ознаки

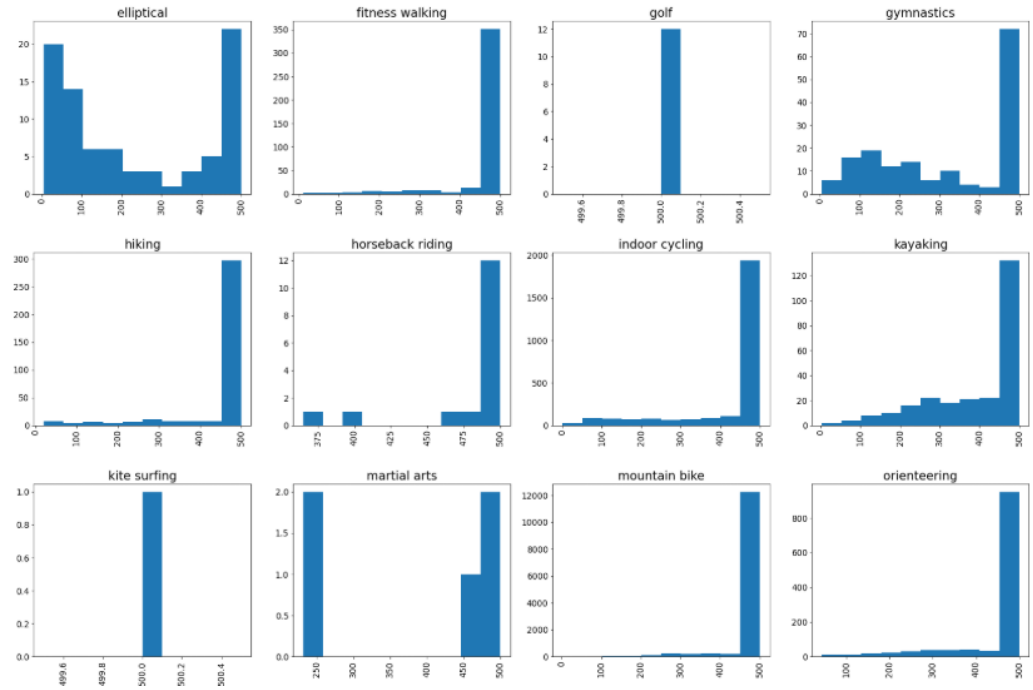


Рисунок 4.8 – Приклад відображення розпорошення даних за значенням цільової ознаки

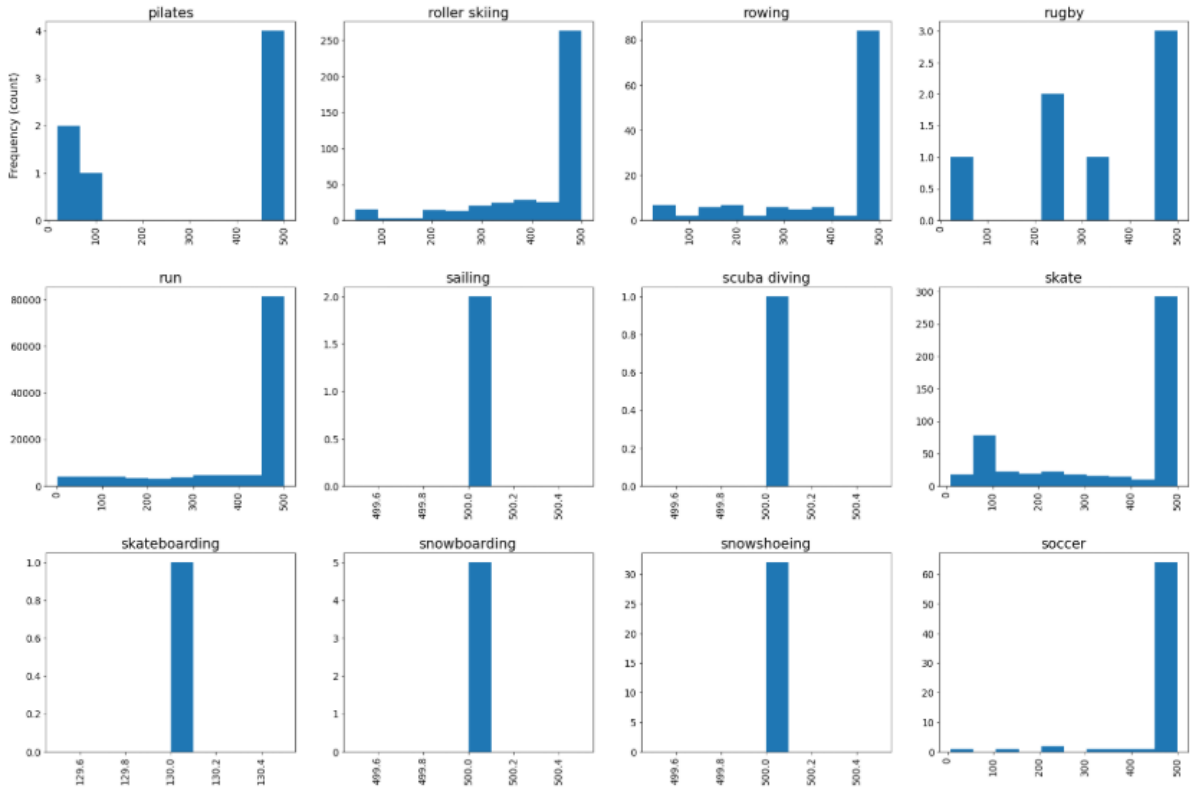


Рисунок 4.9 – Приклад відображення розпорошення даних за значенням цільової ознаки

Результати роботи застосунку наведено у табл. 4.1.

Таблиця 4.1 – Результати тестування застосунку

Час роботи	Послідовне виконання	Паралельне виконання
Час завантаження вибірки	1020 с	22440 с
Час попередньої фільтрації	840 с	780 с
Швидкість візуалізації проміжної інформації	480 с	600 с
Швидкість побудови регресійної моделі	540 с	660 с
Точність отриманої моделі	96%	92%

4.3 Проблеми та виклики

Розроблено застосунок на основі мови програмування Python та фреймворку PySpark. Застосунок дозволяє значно спростити обробку BD. За допомогою багатопочного виконання застосунок дозволяє пришвидшити

виконання обробку BD, але через накладні витрати на пересилання інформації між ядрами прискорення не має лінійного характеру.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було досліджено та розроблено нові засоби обробки великих наборів даних.

Головою метою роботи було прискорення обробки великих наборів даних. Для досягнення цієї мети було використано паралелізацію, засобами фреймворку PySpark, процесів обробки великих наборів даних.

У роботі розроблявся застосунок для обробки великих наборів даних.

В основній частині було представлено дослідницьку частину роботи з результатами огляду найбільш відомих технологій та засобів обробки великих наборів даних. Під час аналізу існуючих рішень було прийнято рішення, щодо розробки нового застосунку, який би вирішував низку проблем існуючих конкурентів.

Для тестування запропонованого рішення використовувалася реальна велика вибірка даних під час опрацювання якої запропонований підхід продемонстрував прийнятні результати, зокрема і прискорення при паралельному виконанні.

За результатами роботи слід зробити висновок про те, що нове рішення обробки великих наборів даних продемонструвало хороші результати.

В якості можливих напрямків розвитку можна розглянути такі варіанти покращення результатів:

– тестування розробленого застосунку на кластерній обпилювальній системі, замість просто багатоядерної обчислювальної системи;

– використання в подальшому мови Java та механізму Java Virtual Machine для оптимізації та пришвидшення роботи застосунку в цілому.

Новизна роботи полягає в тому, що розроблено новий підхід до обробки великих наборів даних.

Практичне значення роботи полягає в тому, що за рахунок використання паралелізації, вдалося пришвидшити обробку великих наборів даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. О'Ніл, Кейт BIG DATA. Зброя математичного знищення. Як великі дані збільшують нерівність і загрожують демократії [Текст] / Кейт О'Ніл ; пер. з англ. Ольга Калініна. - Київ : BookChef : Форс Україна, 2020. - 334 с.

2. Рогушина Ю. В. Дослідження принципів, моделей та методів парадигми менеджменту наукових даних FAIR для аналізу метаданих BIG data [Текст] / Ю. В. Рогушина, І. Ю. Гришанова // Проблеми програмування. - 2021. - № 4. - С. 26-35.

3. Novytskyi O. Methods and techniques for management of ontology-based knowledge representation models in the context of Big data [Текст] / O. Novytskyi // Проблеми програмування. - 2021. - № 4. - С. 19-25.

4. Кучеров Д.П. Методи аналізу великих даних "Big Data" [Текст] : навч. посіб. / Д. П. Кучеров ; Нац. авіац. ун-т. - Київ : НАУ, 2020. - 171 с. :

5. Banyk A. Artificial intelligence techniques for real-time visualisation of big data graph models [Текст] / A. Banyk, P. Mulesa // Інформаційні технології та комп'ютерна інженерія. - 2025. - Т. 22, № 1. - С. 42-54.

6. Одегов М. А. Обґрунтування швидких алгоритмів класифікації на множинах Big Data за критеріями надійності і продуктивності [Текст] / М. А. Одегов, М. М. Гаджиєв, Л. М. Буката, М. В. Глазунова, М. В. Кочеткова // Інфокомунікаційні та комп'ютерні технології. - 2023. - № 1. - С. 148-160.

7. Одегов М. А. Порівняння алгоритмів класифікації Big Data методами імітаційного моделювання [Текст] / М. А. Одегов, М. М. Гаджиєв, Л. М. Буката, М. В. Глазунова, М. В. Кочеткова // Інфокомунікаційні та комп'ютерні технології. - 2023. - № 1. - С. 134-147.

8. Zelenyi D. Using big data for demand forecasting and dynamic pricing [Текст] / D. Zelenyi // Economic forum. - 2025. - Vol. 15, no. 2. - С. 45-55.

9. Nadutenko M. V. The triad "Intellectualization – Virtualization – Big Data": interdisciplinary issues on natural language studies and doctoral education

perspectives [Текст] / M. V. Nadutenko, M. V. Nadutenko, O. L. Fast // Академічні студії. Серія : Педагогіка. - 2024. - Вип. 2. - С. 103-111.

10. Yunis R. Big data analytics for seasonal crop patterns: integrating machine learning techniques [Текст] / R. Yunis, A. Halim, I. A. Pardosi // Eastern-European journal of enterprise technologies. - 2024. - № 6(4). - С. 46–56.

11. Tableau [Electronic resource]. – Access mode : https://www.tableau.com/

12. Інструменти Tableau: повний огляд для бізнесу та аналітики даних [Електрон. ресурс]. – Режим доступу : https://osvita.ua/vnz/student_life/95730/

13. Що таке система Tableau і навіщо вона потрібна аналітикам [Електрон. ресурс]. – Режим доступу : https://profitstore.ua/uk/blog/expert/scho-take-tableau-i-navischo-vona-potribna-analitykam

14. Microsoft Power BI Desktop [Електрон. ресурс]. – Режим доступу : https://www.microsoft.com/uk-ua/download/details.aspx?id=58494

15. Як працювати з Microsoft Power BI — докладний порадник [Електрон. ресурс]. – Режим доступу : https://netpeak.net/uk/blog/yak-pratsyuvati-z-microsoft-power-bi-dokladniy-poradnik/

16. Хмарний сервіс Power BI: ефективність, підтверджена фактами [Електрон. ресурс]. – Режим доступу : https://progresia.online/uk/blog/Khmarnyi-servis-Power-BI-efektyvanist-pidtverdzhena-faktamy

17. RapidMiner Studio [Electronic resource]. – Access mode : https://docs.rapidminer.com/9.9/studio/installation/

18. Altair RapidMiner - платформа для аналізу даних і штучного інтелекту [Електрон. ресурс]. – Режим доступу : https://integralsolutions.pl/uk/altair-rapidminer-ai-%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0-%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7%D1%83-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85/

19. Big Data, Big Opportunities [Текст] / Stefan Wrobel [et al.] // Informatik-Spektrum. – 2014. – Vol. 38, no. 5. – P. 370–378.

20. Mager A. The politics of big data. Big data, big brother? [Текст] / Astrid Mager // Information, Communication & Society. – 2019. – Vol. 22, no. 10. – P. 1523–1525.

21. Viceconti M. Big Data, Big Knowledge: Big Data for Personalized Healthcare [Текст] / Marco Viceconti, Peter Hunter, Rod Hose // IEEE Journal of Biomedical and Health Informatics. – 2015. – Vol. 19, no. 4. – P. 1209–1215.

22. Crop Prediction using PySpark [Текст] / Dr N. Usha Rani [et al.] // International Journal for Research in Applied Science and Engineering Technology. – 2023. – Vol. 11, no. 4. – P. 4298–4302.

23. Mishra R. K. PySpark MLlib and Linear Regression [Текст] / Raju Kumar Mishra // PySpark Recipes. – Berkeley, CA, 2017. – P. 235–259.

24. Singh P. Introduction to Spark [Текст] / Pramod Singh // Learn PySpark. – Berkeley, CA, 2019. – P. 1–16.

25. PySpark Overview [Electronic resource]. – Access mode : https://spark.apache.org/docs/latest/api/python/index.html#pyspark-overview

26. PySpark [Electronic resource]. – Access mode : https://www.databricks.com/glossary/pyspark

27. PyCharm. The Python IDE for data science and web development [Electronic resource]. – Access mode : https://www.jetbrains.com/pycharm/

28. PyCharm як найкраща IDE для розробки ПЗ на Python [Електрон. ресурс]. – Режим доступу : https://foxminded.ua/pycharm-tse/

29. FitRec_Dataset [Electronic resource]. – Access mode : https://www.kaggle.com/datasets/tientd95/fitrec-dataset/data

30. FitRec Datasets [Electronic resource]. – Access mode : https://cseweb.ucsd.edu/~jmcauley/datasets/fitrec.html

ДОДАТОК А

Технічне завдання

Вступ

Дослідження та програмна реалізація засобів обробки великих наборів даних.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація засобів обробки великих наборів даних», затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

А.2 Призначення розробки

Метою роботи є розробка та дослідження нових засобів обробки великих наборів даних.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Під час роботи буде розроблено новий засіб обробки великих наборів даних для. Розглянемо більш детально перелік вимог до нового рішення:

а) обробка великого обсягу даних:

1) повинна підтримувати обробку структурованих, напівструктурованих і неструктурованих даних;

2) повинна обробляти безперервні потоки даних, пакетне завантаження і джерела подій в режимі реального часу.

б) інтеграція з масштабованим сховищем:

1) повинен підключатися до розподілених систем зберігання даних (HDFS, S3, Azure Data Lake, GCS);

2) повинен динамічно масштабуватися зі збільшенням обсягів даних;

3) повинен підтримувати секціонування та індексацію даних для великих наборів даних;

в) механізм розподіленої обробки даних:

1) повинен підтримувати паралельне, розподілене виконання завдань на декількох вузлах;

2) повинен забезпечувати як пакетну, так і потокову обробку даних;

г) відмовостійкість і відновлення:

1) повинен автоматично виявляти збої вузлів і перепризначати завдання;

2) повинен гарантувати одноразову обробку подій;

д) нучке перетворення даних та ETL:

1) повинен підтримувати операції фільтрації, об'єднання, агрегації та управління вікнами;

2) повинен підтримувати еволюцію схеми та робочі процеси очищення даних;

е) розширена аналітика та інтеграція з машинним навчанням:

1) повинні надавати API або модулі для машинного навчання, прогнозування та виявлення аномалій;

2) повинні підтримувати навчання розподілених моделей і їх обслуговування;

є) інформаційна панель і моніторинг в режимі реального часу:

1) необхідно надавати показники продуктивності конвеєра, затримки, пропускної здатності і помилок;

2) необхідно надавати користувачам можливість відстежувати потоки обробки в режимі реального часу;

ж) управління ресурсами та оптимізація:

1) повинен автоматично розподіляти обчислювальні ресурси в залежності від робочого навантаження;

2) повинен підтримувати горизонтальне масштабування в розподілених кластерах;

3) повинен включати плани виконання з урахуванням витрат і продуктивності.

Нові рішення покращать та прискорять обробку великих наборів даних.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс застосунку для обробки великих наборів даних повинен бути ергономічним та зрозумілим для користувачів.

А.3.3 Вимоги до надійності

Вимоги до безпеки та відповідності вимогам – забезпечують дотримання етичних норм і безпеку операцій:

- управління доступом на основі ролей для користувачів;
- знеособлення персональних даних.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для реконструкції тривимірних зображень необхідна наявність персонального комп’ютера у користувача.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для експлуатації програмного забезпечення для реконструкції тривимірних зображень необхідно таке апаратне та програмне забезпечення:

– від 16 ГБ оперативної пам’яті (в двоканальному режимі);

– від 2-х ядер, обчислювальної потужності – 2.0-5.6 ГГц;

– від 100 ГБ жорсткий диск;

– 64-бітна платформа;

– встановлені пакети PySpark та TensorFlow (або наявність безперебійного доступу до мережі).

А.3.6 Вимоги до маркування і пакування

Застосунок для обробки великих наборів даних може бути записаний на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для обробки великих наборів даних».

ДОДАТОК Б

Фрагмент тексту програми

```
from pyspark.sql import SparkSession, DataFrame
from pyspark.sql.types import StructType, StructField, StringType, IntegerType,
TimestampType, DoubleType
from pyspark.sql.functions import col, lit, udf, when, from_json, to_timestamp, expr
from pyspark.sql.window import Window
from pyspark import SparkConf
import pyspark.sql.functions as F
import logging
import json
import os
import uuid
import time
import random
from typing import List, Dict, Any, Optional, Tuple

# ----------------------------------------------------------
# Configuration and Utilities
# ----------------------------------------------------------

LOG_FORMAT = "%(asctime)s - %(levelname)s - %(message)s"
logging.basicConfig(level=logging.INFO, format=LOG_FORMAT)
logger = logging.getLogger("pyspark_big_data_app")

DEFAULT_APP_NAME = "PySparkBigDataApp"
DEFAULT_MASTER = "local[*]"

def create_spark_session(app_name: str = DEFAULT_APP_NAME, master: str =
DEFAULT_MASTER, conf_overrides: Dict[str, str] = None) -> SparkSession:
    """
    Create and return a SparkSession with some default configurations suitable for
development.
    """
    conf = SparkConf().setAppName(app_name).setMaster(master)
    # Default tuning for demo; in production tune according to cluster
    conf.set("spark.sql.shuffle.partitions", "8")
    conf.set("spark.executor.memory", "2g")
    conf.set("spark.sql.adaptive.enabled", "true")
    conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
    if conf_overrides:
        for k, v in conf_overrides.items():
            conf.set(k, v)
    spark = SparkSession.builder.config(conf=conf).getOrCreate()
    logger.info(f"SparkSession created: app={app_name}, master={master}")
    return spark
```

```
def read_json_config(path: str) -> Dict[str, Any]:
    """
    Read JSON configuration file and return dictionary.
    """
    with open(path, "r") as fh:
        cfg = json.load(fh)
    logger.info(f"Loaded configuration from {path}")
    return cfg

def ensure_dir(path: str) -> None:
    if not os.path.exists(path):
        os.makedirs(path, exist_ok=True)
        logger.info(f"Created directory: {path}")

# ---------------------------------------------------------
# Data Source Abstractions
# ---------------------------------------------------------

class DataSource:
    """
    Abstract representation of a data source.
    Types: file, jdbc, kafka, s3, hive
    """
    def __init__(self, source_type: str, options: Dict[str, Any]):
        self.source_type = source_type
        self.options = options

    def read(self, spark: SparkSession) -> DataFrame:
        logger.info(f"Reading from source type: {self.source_type}")
        if self.source_type == "parquet":
            path = self.options.get("path")
            return spark.read.parquet(path)
        elif self.source_type == "csv":
            path = self.options.get("path")
            return spark.read.option("header", "true").csv(path)
        elif self.source_type == "json":
            path = self.options.get("path")
            return spark.read.json(path)
        elif self.source_type == "jdbc":
            return spark.read.format("jdbc").options(**self.options).load()
        elif self.source_type == "kafka":
            return spark.read.format("kafka").options(**self.options).load()
        else:
            raise ValueError(f"Unsupported source type: {self.source_type}")
```

```
# ------------------------------------------------------------
# Data Ingestion
# ------------------------------------------------------------

class DataIngestor:
    def __init__(self, spark: SparkSession, sources: List[DataSource]):
        self.spark = spark
        self.sources = sources

    def ingest_all(self) -> Dict[str, DataFrame]:
        frames = {}
        for idx, src in enumerate(self.sources):
            try:
                df = src.read(self.spark)
                key = f"src_{idx}_{src.source_type}"
                frames[key] = df
                logger.info(f"Ingested source {key} with {df.count()} rows (count triggers
action!)")
            except Exception as e:
                logger.exception(f"Failed to ingest source {idx}: {e}")
        return frames

# ------------------------------------------------------------
# Schema and Data Quality Utilities
# ------------------------------------------------------------

def infer_schema_from_sample(df: DataFrame, max_samples: int = 1000) ->
StructType:
    # Very simple heuristic: use DataFrame.schema directly
    logger.info("Inferring schema from DataFrame")
    return df.schema

def validate_required_columns(df: DataFrame, required: List[str]) -> Tuple[bool,
List[str]]:
    missing = [c for c in required if c not in df.columns]
    ok = len(missing) == 0
    if ok:
        logger.info("All required columns present.")
    else:
        logger.warning(f"Missing required columns: {missing}")
    return ok, missing

# ------------------------------------------------------------
# Preprocessing and Cleaning
```

```
# ------------------------------------------------------------

class Preprocessor:
    def __init__(self, spark: SparkSession):
        self.spark = spark

    def cast_schema(self, df: DataFrame, schema: StructType) -> DataFrame:
        for field in schema.fields:
            if field.name in df.columns:
                df = df.withColumn(field.name, col(field.name).cast(field.dataType))
        return df

    def drop_null_threshold(self, df: DataFrame, threshold: float = 0.3) ->
DataFrame:
        # Drop columns with fraction of nulls > threshold
        total = df.count()
        cols_to_keep = []
        for c in df.columns:
            nulls = df.filter(col(c).isNull()).count()
            frac = nulls / max(total, 1)
            if frac <= threshold:
                cols_to_keep.append(c)
            else:
                logger.info(f"Dropping column {c} due to null fraction {frac:.2f}")
        return df.select(*cols_to_keep)

    def fill_missing(self, df: DataFrame, fill_map: Dict[str, Any]) -> DataFrame:
        return df.fillna(fill_map)

    def deduplicate(self, df: DataFrame, subset: List[str]) -> DataFrame:
        return df.dropDuplicates(subset)

# ------------------------------------------------------------
# Feature Engineering
# ------------------------------------------------------------

class FeatureEngineer:
    def __init__(self, spark: SparkSession):
        self.spark = spark

    def add_time_features(self, df: DataFrame, ts_col: str) -> DataFrame:
        df = df.withColumn("hour_of_day", F.hour(col(ts_col))) \
               .withColumn("day_of_week", F.dayofweek(col(ts_col))) \
               .withColumn("month_of_year", F.month(col(ts_col)))
        return df
```

```
    def encode_categorical(self, df: DataFrame, col_name: str, prefix: str = "cat") ->
DataFrame:
        # naive one-hot via pivot; in real life use StringIndexer + OneHotEncoder
        vals = [r[col_name] for r in df.select(col_name).distinct().collect()]
        for v in vals:
            safe_v = str(v).replace(" ", "_").replace("/", "_")
            df = df.withColumn(f"{prefix}_{safe_v}", when(col(col_name) == v,
1).otherwise(0))
        return df

    def aggregate_by_key(self, df: DataFrame, key: str, agg_exprs: Dict[str, str]) ->
DataFrame:
        agg_list = []
        for cn, fn in agg_exprs.items():
            if fn.lower() == "sum":
                agg_list.append(F.sum(col(cn)).alias(f"{cn}_sum"))
            elif fn.lower() == "avg":
                agg_list.append(F.avg(col(cn)).alias(f"{cn}_avg"))
            elif fn.lower() == "max":
                agg_list.append(F.max(col(cn)).alias(f"{cn}_max"))
            elif fn.lower() == "min":
                agg_list.append(F.min(col(cn)).alias(f"{cn}_min"))
            elif fn.lower() == "count":
                agg_list.append(F.count(col(cn)).alias(f"{cn}_count"))
            else:
                # default: count
                agg_list.append(F.count(col(cn)).alias(f"{cn}_count"))
        return df.groupBy(key).agg(*agg_list)

# ------------------------------------------------------------
# Machine Learning (MLlib) Utilities
# ------------------------------------------------------------

from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler, StringIndexer
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

class ModelTrainer:
    def __init__(self, spark: SparkSession):
        self.spark = spark
```

```
    def build_classification_pipeline(self, feature_cols: List[str], label_col: str) ->
Pipeline:
        assembler = VectorAssembler(inputCols=feature_cols,
outputCol="features_raw")
        scaler = StandardScaler(inputCol="features_raw", outputCol="features")
        lr = LogisticRegression(featuresCol="features", labelCol=label_col)
        pipeline = Pipeline(stages=[assembler, scaler, lr])
        return pipeline

    def train_with_cv(self, df: DataFrame, pipeline: Pipeline, param_grid: Dict[str,
List[Any]], label_col: str, num_folds: int = 3) -> Any:
        evaluator = BinaryClassificationEvaluator(labelCol=label_col,
rawPredictionCol="rawPrediction")
        pg_builder = ParamGridBuilder()
        for param, vals in param_grid.items():
            pg_builder = pg_builder.addGrid(param, vals)
        pg = pg_builder.build()
        cv = CrossValidator(estimator=pipeline, estimatorParamMaps=pg,
evaluator=evaluator, numFolds=num_folds)
        model = cv.fit(df)
        logger.info("Completed cross-validated training")
        return model

    def evaluate(self, model: Any, df: DataFrame, label_col: str) -> Dict[str, float]:
        preds = model.transform(df)
        evaluator = BinaryClassificationEvaluator(labelCol=label_col)
        auc = evaluator.evaluate(preds)
        # some additional metrics
        tp = preds.filter((col(label_col) == 1) & (col("prediction") == 1)).count()
        fp = preds.filter((col(label_col) == 0) & (col("prediction") == 1)).count()
        fn = preds.filter((col(label_col) == 1) & (col("prediction") == 0)).count()
        tn = preds.filter((col(label_col) == 0) & (col("prediction") == 0)).count()
        acc = (tp + tn) / max((tp + fp + fn + tn), 1)
        metrics = {"auc": auc, "accuracy": acc, "tp": tp, "fp": fp, "fn": fn, "tn": tn}
        logger.info(f"Evaluation metrics: {metrics}")
        return metrics

# ------------------------------------------------------------
# Streaming Utilities (Structured Streaming)
# ------------------------------------------------------------

class StreamProcessor:
    def __init__(self, spark: SparkSession, checkpoint_dir: str = "/tmp/checkpoints"):
        self.spark = spark
        self.checkpoint_dir = checkpoint_dir
```

```
        ensure_dir(self.checkpoint_dir)

    def read_from_kafka(self, options: Dict[str, str]) -> DataFrame:
        return self.spark.readStream.format("kafka").options(**options).load()

    def parse_json_value(self, df: DataFrame, schema: StructType, value_col: str =
"value") -> DataFrame:
        parsed = df.select(from_json(col(value_col).cast("string"),
schema).alias("data"))
        exploded = parsed.select("data.*")
        return exploded

    def write_to_console(self, df: DataFrame, query_name: str = "stream_console") -
> None:
        q = df.writeStream.format("console").option("truncate",
"false").option("numRows", 20).start()
        logger.info(f"Started console stream query: {query_name}")
        q.awaitTermination(timeout=5)
        q.stop()

    def simple_stream_aggregate(self, df: DataFrame, key_col: str, time_col: str,
window_duration: str = "1 minute") -> Any:
        # expects timestamp column time_col
        grouped = df.withWatermark(time_col, "2 minutes") \
            .groupBy(F.window(col(time_col), window_duration), col(key_col)) \
            .count()
        q =
grouped.writeStream.format("console").outputMode("update").option("truncate",
"false").start()
        logger.info("Started stream aggregate query")
        q.awaitTermination(timeout=5)
        q.stop()

# ------------------------------------------------------------
# Storage / Output Management
# ------------------------------------------------------------

class StorageManager:
    def __init__(self, spark: SparkSession):
        self.spark = spark

    def write_parquet(self, df: DataFrame, path: str, mode: str = "overwrite") ->
None:
        logger.info(f"Writing DataFrame to Parquet: {path}")
        df.write.mode(mode).parquet(path)
```

```
    def write_csv(self, df: DataFrame, path: str, mode: str = "overwrite") -> None:
        logger.info(f"Writing DataFrame to CSV: {path}")
        df.write.mode(mode).option("header", "true").csv(path)

    def write_to_jdbc(self, df: DataFrame, options: Dict[str, Any]) -> None:
        logger.info("Writing DataFrame to JDBC sink")
        df.write.format("jdbc").options(**options).mode("append").save()

# ---------------------------------------------------------
# Monitoring and Metrics
# ---------------------------------------------------------

class MetricsCollector:
    def __init__(self):
        self.metrics = {}

    def gauge(self, name: str, value: float) -> None:
        self.metrics[name] = value
        logger.info(f"Metric gauge: {name} = {value}")

    def increment(self, name: str, delta: int = 1) -> None:
        self.metrics[name] = self.metrics.get(name, 0) + delta
        logger.info(f"Metric increment: {name} -> {self.metrics[name]}")

    def get_all(self) -> Dict[str, Any]:
        return dict(self.metrics)

# ---------------------------------------------------------
# Example Application Flow
# ---------------------------------------------------------

def example_batch_workflow(spark: SparkSession, input_path: str, output_path: str)
-> None:
    # data ingestion
    ds = DataSource("parquet", {"path": input_path})
    df_raw = ds.read(spark)

    pre = Preprocessor(spark)
    df_clean = pre.drop_null_threshold(df_raw, threshold=0.5)
    df_clean = pre.deduplicate(df_clean, subset=["id"])

    fe = FeatureEngineer(spark)
    if "timestamp" in df_clean.columns:
        df_fe = fe.add_time_features(df_clean, "timestamp")
```

```
    else:
        df_fe = df_clean

    # prepare ML dataset
    feature_cols = [c for c in df_fe.columns if c not in ("label", "id", "timestamp")]
    label_col = "label" if "label" in df_fe.columns else None
    if label_col:
        trainer = ModelTrainer(spark)
        pipeline = trainer.build_classification_pipeline(feature_cols, label_col)
        # dummy param grid
        params = { "stages_2_regParam": [0.01, 0.1] } if hasattr(pipeline, "stages") else
{}
        try:
            # in real life df needs train/test split and vector features
            model = pipeline.fit(df_fe)
            metrics = trainer.evaluate(model, df_fe, label_col)
            logger.info(f"Model metrics: {metrics}")
        except Exception as e:
            logger.exception("Training failed", exc_info=e)

    storage = StorageManager(spark)
    storage.write_parquet(df_fe, os.path.join(output_path, "processed"))

# ---------------------------------------------------------
# Example: Complex Data Pipeline with Multiple Steps
# ---------------------------------------------------------

def complex_pipeline_example(spark: SparkSession, cfg: Dict[str, Any]) -> None:
    # cfg includes sources, preprocess rules, features, ml params, outputs
    sources_cfg = cfg.get("sources", [])
    sources = []
    for s in sources_cfg:
        sources.append(DataSource(s.get("type"), s.get("options", {})))

    ingestor = DataIngestor(spark, sources)
    frames = ingestor.ingest_all()

    pre = Preprocessor(spark)
    fe = FeatureEngineer(spark)
    storage = StorageManager(spark)
    metrics = MetricsCollector()

    for name, df in frames.items():
        logger.info(f"Processing frame: {name}")
        # sample and schema inference
```

```
        schema = infer_schema_from_sample(df)
        df_cast = pre.cast_schema(df, schema)
        df_clean = pre.drop_null_threshold(df_cast, threshold=0.4)
        df_clean = pre.deduplicate(df_clean, subset=[c for c in df_clean.columns if
c.endswith("_id")][:1] or ["id"])
        # feature transformations
        if "timestamp" in df_clean.columns:
            df_feat = fe.add_time_features(df_clean, "timestamp")
        else:
            df_feat = df_clean
        # write intermediate
        out_path = cfg.get("output_base", "/tmp/output")
        storage.write_parquet(df_feat, os.path.join(out_path, name))
        metrics.gauge(f"{name}_rows", float(df_feat.count()))

# ------------------------------------------------------------
# Orchestration Entry Point
# ------------------------------------------------------------

def main(argv: Optional[List[str]] = None) -> None:
    # parse args rudimentary
    spark = create_spark_session()
    try:
        # mocked config for the example
        cfg = {
            "sources": [
                {"type": "parquet", "options": {"path": "/path/to/input1"}},
                {"type": "csv", "options": {"path": "/path/to/input2"}},
            ],
            "output_base": "/tmp/pyspark_app_out"
        }
        complex_pipeline_example(spark, cfg)
        # demonstration of batch workflow
        example_batch_workflow(spark, "/path/to/example/parquet",
"/tmp/pyspark_example_output")
    except Exception as e:
        logger.exception("Application run failed", exc_info=e)
    finally:
        spark.stop()
        logger.info("SparkSession stopped")

# ------------------------------------------------------------
# Additional Helpers and Mock Components (to increase content)
# ------------------------------------------------------------
```

```
def generate_synthetic_data(spark: SparkSession, n: int = 1000) -> DataFrame:
    # create a DataFrame with synthetic data for testing
    rows = []
    for i in range(n):
        rows.append((str(uuid.uuid4()), i % 100, time.time() + i,
float(random.random())))
    rdd = spark.sparkContext.parallelize(rows)
    schema = StructType([
        StructField("id", StringType(), True),
        StructField("user_id", IntegerType(), True),
        StructField("timestamp", DoubleType(), True),
        StructField("value", DoubleType(), True),
    ])
    df = spark.createDataFrame(rdd, schema=schema)
    df = df.withColumn("timestamp", to_timestamp(col("timestamp")))
    return df

def example_using_generated_data(spark: SparkSession) -> None:
    df = generate_synthetic_data(spark, n=500)
    pre = Preprocessor(spark)
    df = pre.fill_missing(df, {"value": 0.0})
    fe = FeatureEngineer(spark)
    df = fe.add_time_features(df, "timestamp")
    storage = StorageManager(spark)
    out = "/tmp/synthetic_out"
    storage.write_parquet(df, out)

# ------------------------------------------------------------
# Misc: Utilities for UDF examples and Pandas UDF placeholders
# ------------------------------------------------------------

@udf(returnType=DoubleType())
def multiply_by_two(x):
    try:
        return float(x) * 2.0
    except Exception:
        return None

def register_example_udfs(spark: SparkSession) -> None:
    spark.udf.register("multiply_by_two", multiply_by_two)
    logger.info("Registered example UDFs")

# ------------------------------------------------------------
# CLI guard
# ------------------------------------------------------------
```

```
if __name__ == "__main__":
    main()
```

ДОДАТОК В

Слайди презентації

Міністерство освіти та науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дослідження та програмна реалізація засобів обробки великих наборів даних

Виконав: студент групи КНТ-214м **Ігорь МУРАВСЬКИЙ**
Керівник: д-р техн. наук, професор **Михайло ПОЛЯКОВ**

Рисунок В.1 – Слайд 1

ОБ'ЄКТ, ПРЕДМЕТ ТА МЕТА РОБОТИ

- **Об'єкт дослідження – процес обробки великих наборів даних.**
- **Предмет дослідження – методи та засоби для обробки великих наборів даних.**
- **Метою роботи є прискорення процесу обробки великих наборів даних, шляхом паралелізації.**

2

Рисунок В.2 – Слайд 2

АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

	Tableau	Microsoft Power BI	RapidMiner
Основне призначення	Візуалізація та аналітика даних	Бізнес-звітність та аналітика	Наука про дані, машинне навчання та аналітика
Основні недоліки	Висока вартість ліцензування; обмежений обсяг розширеної аналітики; зниження продуктивності при роботі з дуже великими наборами даних в режимі реального часу; для оптимізації потрібні досвідчені користувачі; немає вбудованого машинного навчання	Продуктивність обмежена через обмеження набору даних; складна крива навчання DAX; DirectQuery може працювати повільно; обмежена при використанні поза Microsoft stack; обмеження пам'яті в настільній версії	Високе споживання ресурсів; вимагає потужних машин; висока швидкість навчання; обмежена візуалізація; дорога корпоративна версія; не підходить для простих інформаційних панелей
Обробка великих обсягів даних	Середньо-висока (залежить від ресурсів сервера та вилучення даних)	Середньо-висока (добре працює зі службами Azure; слабка автономність)	Висока продуктивність (підтримує середовища Hadoop / Spark)
Можливості інтеграції	Багато з'єднувачів (Hadoop, Spark, SQL, хмарні сховища)	Чудово працює з продуктами Microsoft; багато зовнішніх з'єднань	Інтегрується з Hadoop, Spark, хмарними сховищами даних, базами даних SQL
Масштабованість	Масштабується за допомогою сервера Tableau / Cloud, але дорого	Добре масштабується тільки в рамках екосистеми Microsoft	Добре масштабується в корпоративних версіях і гірше в безкоштовній / настільній версії
Ідеально підходить для	Підприємства віддають перевагу візуалізації даних та інформаційним панелям BI	Організації, що використовують Microsoft stack та економічні робочі процеси BI	Спеціалістам з обробки даних потрібні наскрізні робочі процеси ML

3

Рисунок В.3 – Слайд 3

ЗАВДАННЯ РОБОТИ

Метою роботи є розробка та дослідження нових засобів обробки великих наборів даних.

Під час роботи буде розроблено новий засіб обробки великих наборів даних , з використанням паралелізації , для прискорення процесу обробки.

Розроблення нового засобу обробки великих наборів даних дозволить:

- завантажувати та розгортати великі набори даних в реальному часі;
- прискорити обробку таких великих наборів даних;
- налаштовувати паралельну обробку даних.

4

Рисунок В.4 – Слайд 4

ВИБІР ТЕХНОЛОГІЇ

	Hadoop (MapReduce)	Apache Spark	Apache Flink
Модель первинної обробки	Пакетна обробка	Єдина пакетна і мікропакетна потокова передача даних	Потокове передавання в реальному часі та пакетна обробка
Основні переваги	Висока надійність; обробляє величезні обсяги даних; проста масштабована архітектура	Надзвичайно швидка обробка даних; обчислення в оперативній пам'яті; багата екосистема	Краща продуктивність у режимі реального часу; керована подіями
Слабкість	Повільність при виконанні ітеративних завдань; висока затримка; опеації з великим обсягом сховища	Високі вимоги до пам'яті; мікропакетна обробка даних може обмежувати точність у режимі реального часу	Більш складна екосистема
Типові варіанти використання	ETL, масштабні пакетні завдання, обробка журналів	Машинне навчання, аналітика, обробка графіків, мікропакетне потокове передавання даних	Аналітика в реальному часі, обробка подій, безперервні конвеєри
Тип затримки	Висока затримка (хвилини / години)	Низька / середня затримка (секунди)	Дуже низька затримка (мілісекунди)
Масштабованість	Дуже висока	Дуже висока	Дуже висока
Мова програмування	Java, Python, R та інші за допомогою потокової передачі даних Hadoop	Scala, Java, Python, R.	Java, Scala, Python
Складність моделі	Проста модель програмування	Помірна складність; багатий набір API	Більш висока складність, потокова підтримка
Розгортання	Локальна або хмарна	Локальний або хмарний	Локальна або хмарна
Найкраще для	Застарілі робочі навантаження з великими обсягами даних	Швидка аналітика, загальні робочі процеси з великими даними	Критично важливе потокове передавання в режимі реального часу

5

Рисунок В.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ

	Python	Scala	Java
Основні переваги	Простота написання; багата екосистема машинного навчання; гнучкість; швидке створення прототипів	Рідна мова для Spark; висока продуктивність ; функціональність ; компілюється	Стабільність ; висока продуктивність
Рівень продуктивності	Середнє (інтерпретується, більш повільне виконання в початковому вигляді)	Швидко (працює на JVM, оптимізовано для Spark)	Від швидкого до надшвидкого (оптимізовано для JVM, потужний компілятор JIT)
Поріг входження в розробку (складність вивчення)	Низький (легко вивчається)	Високий (складна у вивченні)	Середній (може бути адаптовано під попередній досвід)
Екосистема для великих даних	Багато бібліотек та додаткових фреймворків	Середня кількість бібліотек та додаткових фреймворків	Середня кількість бібліотек та додаткових фреймворків
Найкращі варіанти використання	Обробка даних, конвеєри на основ ML, оркестровка, прототипування	Інструменти потокової передачі в реальному часі, розподілені двигуни, рідні програми Spark	Пакетні двигуни, розподілені системи низького рівня, корпоративні платформи обробки великих даних
Типова швидкість розвитку	Дуже швидкий (висока продуктивність)	Помірна (більш складна в розробці)	Повільніше (більше шаблонів)
Ефективність використання пам'яті	Висока	Висока	Висока
Підтримка спільноти	Потужна спільнота	Спільнота розвинена, але не активна	Спільнота розвинена, але не активна

6

Рисунок В.6 – Слайд 6

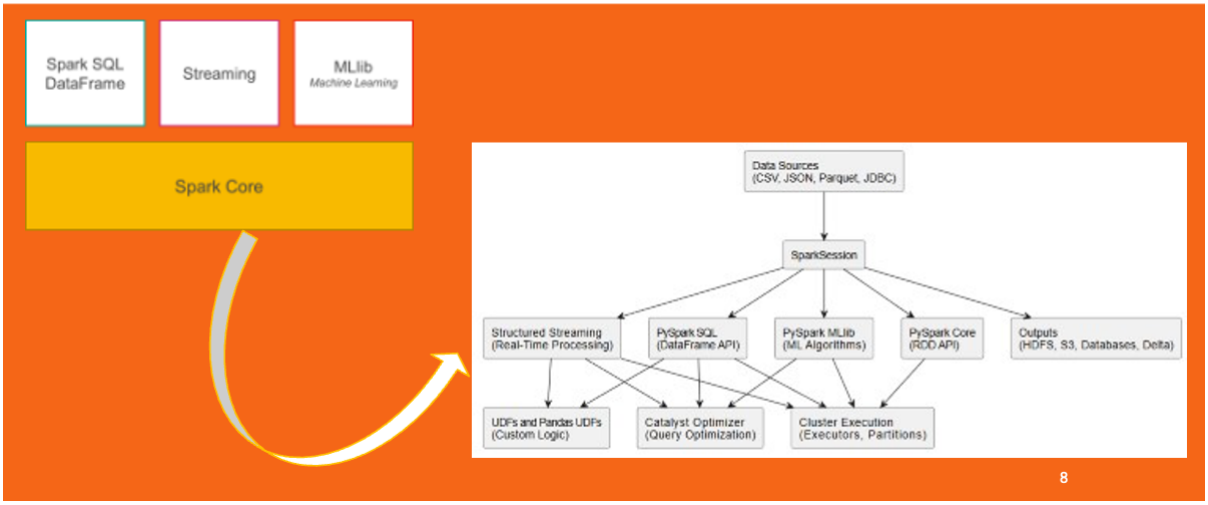
ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

Критерій порівняння мов програмування	Visual Studio	PyCharm
Підтримки Python	Може бути адаптована за допомогою надбудов та плагінів	Спеціалізоване IDE під Python
Простот а використання	Загалом зручне IDE, але не зручне для розробки масштабованих застосунків	Оптимізовано під розробку будь -яких застосунків на Python
Кросплатформність	Windows та macOS; Linux – обмежено	Доступно для Windows, macOS і Linux.
Адаптивність середовища	Ресурсоємність збільшується при збільшенні сторонніх модулів	Легший та оптимізованіший за Visual Studio з набором плагінів, але може трішки сповільнятися при роботі з великими проєктами
Навігація по коду	Зручна навігація, розширена: IntelliSense для Python	Повністю адаптовано під Python
Розгортання фреймворків	Підтримує тестування на Python, але для інших надбудов потрібні додаткові налаштування	Підтримка фреймворків тестування за замовченням (unittest, pytest та nose)
Розширення / плагіни	Великий вибір сторонніх бібліотек та модулів (але може сповільнити завантаження)	Велика кількість перевірених сторонніх бібліотек та модулів
Управління віртуальним середовищем	Реалізовані тільки базові механізми	Автоматизоване визначення та налаштування віртуальних середовищ
Підтримка наукових досліджень даних	Є інтеграція з Jupyter Notebooks, але загалом не розраховано на наукові дослідження	Налаштована потужна інтеграція з безліччю сторонніх надбудов для дослідження даних

7

Рисунок В.7 – Слайд 7

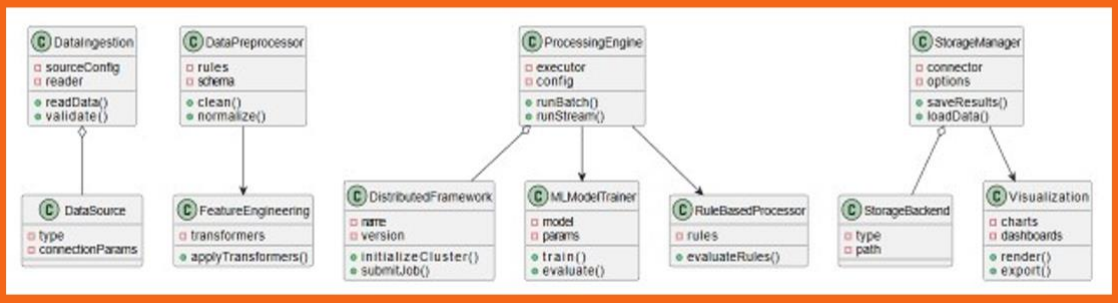
ЗАГАЛЬНА СХЕМА РОБОТИ – ВИКОРИСТАННЯ КОМПОНЕНТІВ PYSPARK



8

Рисунок В.8 – Слайд 8

ФІЗИЧНЕ ПРОЄКТУВАННЯ



9

Рисунок В.9 – Слайд 9

РЕЗУЛЬТАТИ ТЕСТУВАННЯ – ЗАВАНТАЖЕННЯ ДАНИХ

	summary	gender	id	sport	url	userId
0	count	253020	253020	253020	253020	253020
1	mean	None	3.5662444129261328	None	None	4619648.939783417
2	stddev	None	1.5748456348953188	None	None	3932877.7296880507
3	min	female	99296	aerobics	https://www.endomondo.com/users/10014612/worko...	69
4	max	unknown	674008008	yoga	https://www.endomondo.com/users/9991401/workou...	15481421

```
There are total 253020 row, Let print first 2 data rows:
```

	altitude	gender	heart_rate	id	latitude	longitude	speed	sport	timestamp	url	userId
0	[41.6, 40.6, 40.6, 38.4, 37.0, 34.0, 34.0, 34....	male	[100, 111, 120, 119, 120, 116, 125, 128, 131, ...	396826535	[60.173348765820265, 60.173239801079035, 60.17...	[24.64977040886879, 24.65014273300767, 24.6509...	[6.8652, 16.4736, 19.1988, 20.4804, 31.3956, 3...	bike	[1408898746, 1408898754, 1408898765, 1408898877...	https://www.endomondo.com/users/10921915/worko...	10921915
1	[38.4, 39.0, 39.0, 38.2, 36.8, 36.8, 36.8, 35....	male	[100, 105, 111, 110, 108, 115, 126, 130, 132, ...	392337038	[60.173247596248984, 60.17320962622762, 60.172...	[24.649855233728886, 24.65015547350049, 24.650...	[9.0792, 13.284, 15.9336, 10.9476, 16.1676, 30...	bike	[1408221682, 1408221687, 1408221699, 1408221700...	https://www.endomondo.com/users/10921915/worko...	10921915

10

Рисунок В.10 – Слайд 10

Рисунок В.11 – Слайд 11

Час роботи	Послідовне виконання	Паралельне виконання
Час завантаження вибірки	1020 с	22440 с
Час попередньої фільтрації	840 с	780 с
Швидкість візуалізації проміжної інформації	480 с	600 с
Швидкість побудови регресійної моделі	540 с	660 с
Точність отриманої моделі	96%	92%

Рисунок В.12 – Слайд 12

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було досліджено та розроблено нові засоби обробки великих наборів даних.

Головою метою роботи було прискорення обробки великих наборів даних. Для досягнення цієї мети було використано паралелізацію, засобами фреймворку PySpark, процесів обробки великих наборів даних.

Для тестування запропонованого рішення використовувалася реальна велика вибірка даних під час опрацювання якої запропонований підхід продемонстрував прийнятні результати, зокрема і прискорення при паралельному виконанні.

Новизна роботи полягає в тому, що розроблено новий підхід до обробки великих наборів даних.

Практичне значення роботи полягає в тому, що за рахунок використання паралелізації, вдалося пришвидшити обробку великих наборів даних.

13

Рисунок В.13 – Слайд 13