МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра системного аналізу та управління

К.С. Хабарлак

АНАЛІЗ ТА ОБРОБКА ВЕЛИКИХ ДАНИХ

Методичні рекомендації до виконання практичних робіт
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Дніпро
НТУ «ДП»
2024

Хабарлак К.С.

Аналіз та обробка великих даних [Електронний ресурс] : методичні рекомендації до виконання практичних робіт для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 82 с.

Автор:

Хабарлак К.С., доктор філософії.

Затверджено науково-методичною комісією зі спеціальності 124 Системний аналіз (протокол № 3 від 10.05.2024) за поданням кафедри системного аналізу та управління (протокол № 5 від 10.05.2024).

Методичні рекомендації мають на меті допомогти студентам спеціальності 124 Системний аналіз у самостійному засвоєнні обов’язкової дисципліни «Аналіз та обробка великих даних» під час виконання практичних робіт.

Розглянуто теоретичні засади систем розподіленої обробки даних та нереляційних баз даних. Описано процедуру конфігурації, типові запити та функції таких систем. Наведено приклади розв’язку практичних задач за наведеними алгоритмами.

Сформульовано вимоги до оформлення звіту до практичних робіт, питання для самоконтролю та критерії оцінювання практичних робіт.

Рекомендації орієнтовано на активізацію виконавчого етапу навчальної діяльності студентів.

Відповідальний за випуск завідувач кафедри системного аналізу та управління Т.А. Желдак, канд. техн. наук, доц.

Зміст

Вступ

Вміння аналізувати великі дані є затребуваною навичкою, адже людина, пристрої Інтернету речей постійно генерують неосяжні об’єми даних. В необробленому вигляді вони мають невисоку цінність, так як їх складно зберігати, структурувати та аналізувати, тому все більший розвиток отримують підходи аналізу та обробки великих даних**.**

У даному посібнику подано методичні рекомендації щодо виконання практичних робіт з курсу «Аналіз та обробка великих даних». Завдання, викладені у даному посібнику, призначені для того, щоб надати студентам практичний досвід конфігурації та застосування систем розподіленої обробки та зберігання даних, виконання запитів до них, перетворення даних та використання моделей машинного навчання.

Завдяки цим завданням здобувач отримає практичні навички роботи з інструментарієм обробки та аналізу великих даних: бібліотеками Apache Spark і PySpark, нереляційною базою даних MongoDB, мовою програмування Python для аналізу великих даних. Здобувач навчиться збирати, зберігати, оброблювати та аналізувати великі масиви даних; будувати моделі регресії та класифікації, використовуючи великі набори даних, та робити передбачення на нових, невідомих вхідних значеннях; опанує нереляційні бази даних для зберігання та обробки великих даних.

Мета дисципліни – сформувати у здобувачів вищої освіти навички збору, обробки та аналізу великих із використанням сучасних бібліотек Apache Spark і MongoDB та мови програмування Python. Знання та навички, отримані в курсі, будуть корисними для подальшого працевлаштування здобувача.

Завдання курсу:

- навчитися збирати, зберігати, оброблювати та аналізувати великі масиви даних;
- навчитися будувати моделі регресії та класифікації, використовуючи великі набори даних, та робити передбачення на нових, невідомих вхідних значеннях;
- опанувати нереляційні бази даних для зберігання та обробки великих даних;
- отримати практичні навички роботи з бібліотеками Apache Spark, PySpark, базою даних MongoDB, мовою програмування Python для обробки та аналізу великих даних.

Дисциплінарні результати навчання:

1. Здійснювати пошук, агрегацію та візуалізацію великих даних для аналізу поведінки досліджуваної системи.
2. Вміти застосовувати методи побудови регресії на великих даних. Проводити аналіз та прогнозування.

3. Вміти застосовувати методи класифікації (лінійних, метричних моделей та на основі дерев рішень) із використанням інструментарію обробки великих даних мови програмування Python.
4. Розуміти способи підгонки параметрів моделей машинного навчання на великих даних.
5. Вміти проводити первинну обробку великих масивів даних різної природи, виявляти закономірності та візуалізувати дані.
6. Розуміти підходи щодо обробки та аналізу слабко структурованих даних для зберігання в нереляційних базах даних, вміти проводити їх подальший аналіз.

Практична робота №1 – Джерела великих даних. Обробка слабко-структурованої інформації різного походження

Мета роботи: закріплення навичок роботи із кадрами даних PySpark, початковий аналіз даних, візуалізація великих даних.

Практична робота присвячена знайомству із фреймворком розподіленої обробки даних Apache Spark. В ході практичної роботи здобувач налаштує середовище для виконання програм Apache Spark на власному комп'ютері та навчиться завантажувати та писати запити до даних із використанням інтерфейсу Apache Spark до мови програмування Python PySpark. Для розв'язання задачі практичної роботи за узгодженням з викладачем студент може запропонувати свій набір даних.

1.1. Теоретичні відомості

Що таке Spark? Spark – це потужний аналітичний механізм для обробки великих даних, спрямований на швидкість, простоту використання та розширюваність для програм великих даних. Сьогодні Spark став де-факто стандартом для структурування та керування додатками великих даних. Це перевірена та широко поширена технологія, яку використовують багато компаній, які щодня обробляють великі дані.

Spark надає API високого рівня в Java, Python, Scala і R. У даному посібнику ми будемо користуватись Python через PySpark, API, який надає модель програмування Spark для Python. Оскільки Python є найдоступнішою мовою програмування для потужного і експресивного API Spark, простота PySpark робить його найкращим вибором для нас. PySpark надає такі дві важливі функції:

- він дозволяє нам писати програми Spark за допомогою API Python;
- він забезпечує оболонку PySpark для інтерактивного аналізу даних у розподіленому середовищі.

Apache Spark має низку функцій, яких не було у його попередників, наприклад здатність до обробки в пам'яті та потокової обробки, але найважливішою особливістю є те, що Spark є єдиною платформою для обробки великих даних.

Це означає, що Spark поставляється з кількома вбудованими бібліотеками та інструментами, які стосуються різних аспектів роботи з великими даними. Він має вбудований механізм роботи з SQL для виконання обробки великих даних, бібліотеку для масштабованого машинного навчання, механізм потокової обробки для потокової аналітики та багато іншого.

Великі компанії мають багато різноманітних потреб у даних, і, як наслідок, інженерам і аналітикам, доводиться об'єднувати та інтегрувати багато інструментів і методів разом, щоб вони могли створювати різні конвеєри даних для задоволення цих потреб. Але такий підхід може створити дуже серйозну проблему залежності, що створить велику перешкоду для підтримки цього

робочого процесу. Це одна з головних причин, чому Spark досяг такого успіху – він вже містить майже все, що вам може знадобитися.

Загальна структура Spark. Якщо у вас є невеликі дані, їх можна проаналізувати за допомогою одного комп'ютера за розумний проміжок часу. Якщо у вас є великі обсяги даних, використання одного комп'ютера для аналізу й обробки даних (і їх зберігання) може бути надто повільним або навіть неможливим. Ось чому ми хочемо використовувати Spark.

Spark має основну бібліотеку та набір вбудованих бібліотек (SQL, GraphX, Streaming, MLlib), як показано на рисунку 1.1. Як можна побачити, за допомогою DataSource API Spark може взаємодіяти з багатьма джерелами даних, такими як MongoDB, Hadoop, HBase, Amazon S3, Elasticsearch, MySQL та інші.

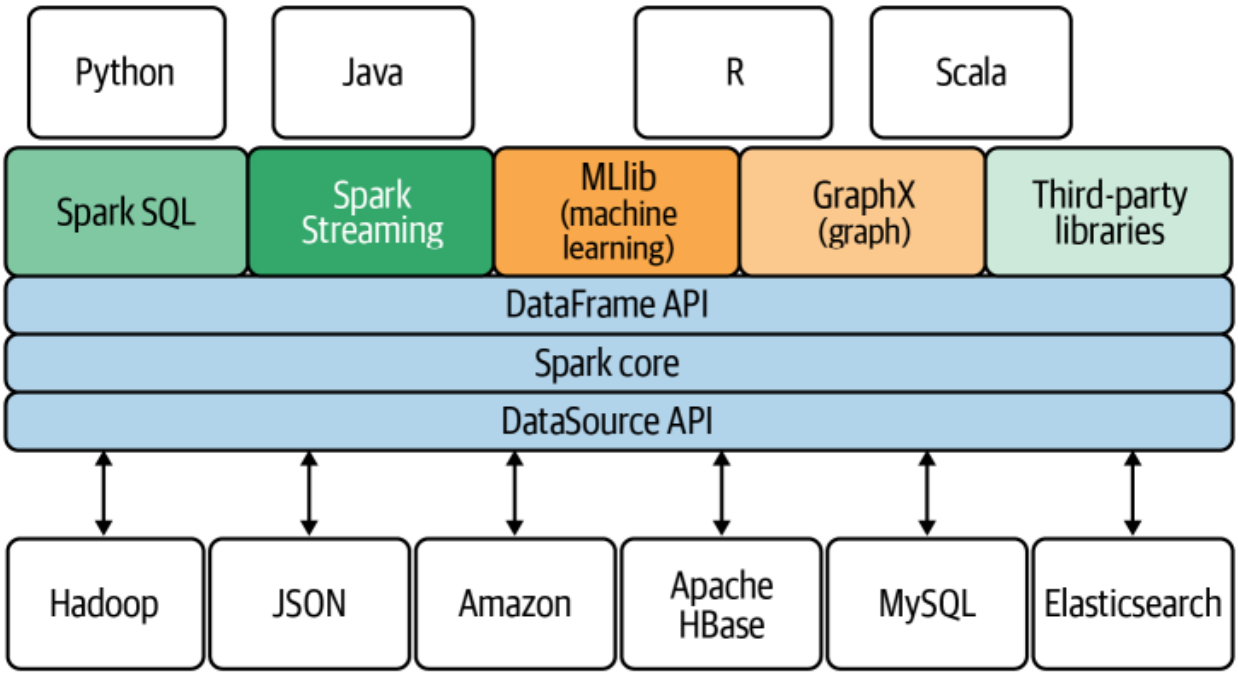

Рисунок 1.1. Бібліотеки Apache Spark.

Цей рисунок показує справжню потужність Spark: можливо використати кілька різних мов для написання програм Spark, а потім застосувати багатий набір бібліотек для вирішення різноманітних проблем з великими даними. При цьому читання та запис можна здійснювати з різних джерел даних.

Ключові терміни Apache Spark. Щоб зрозуміти архітектуру Spark, необхідно зрозуміти кілька ключових термінів:

– *SparkSession* (сесія Spark). Клас SparkSession, визначений у пакеті pyspark.sql, є точкою входу для програмування Spark за допомогою API наборів даних (Dataset) і кадрів даних (DataFrame). Щоб зробити щось корисне з кластером Spark, вам спочатку потрібно створити екземпляр цього класу, який надає вам доступ до екземпляра SparkContext.

- *SparkContext* (контекст Spark). Клас SparkContext, визначений у пакеті pyspark, є основною точкою входу для функціональності Spark. SparkContext підтримує з'єднання з менеджером кластера Spark і може використовуватися для створення так званих стійких розподілених наборів даних (Resilient Distributed Dataset, RDD) і широкомовних змінних у кластері. Коли ви створюєте екземпляр SparkSession, SparkContext стає доступним у вашому сеансі як атрибут SparkSession.sparkContext.
- *Driver* (драйвер). Усі програми Spark (включаючи ті, що використовують оболонку PySpark та автономні програми Python) працюють як незалежні набори процесів. Ці процеси координуються контекстом Spark у програмі драйвера. Щоб надіслати автономну програму Python до Spark, необхідно написати програму драйвера із використанням PySpark API (або Java, або Scala). Ця програма відповідає за процес запуску функції main() програми та створення SparkContext. Його також можна використовувати для створення RDD і DataFrame.
- *Worker* (робітник). У кластерному середовищі Spark є два типи вузлів: один (або два, для високої доступності) головні *(master)* і набір робочих. Робочий – будь-який вузол, який може запускати програми в кластері. Якщо для програми запущено процес, ця програма отримує виконавців на робочих вузлах, які відповідають за виконання завдань Spark.
- *Cluster manager* (менеджер кластера). Master вузол також відомий як менеджер кластера. Основною функцією цього вузла є керування середовищем кластера та серверами, які Spark використовуватиме для виконання завдань. Менеджер кластера розподіляє ресурси для кожної програми. Spark підтримує п'ять типів менеджерів кластерів, залежно від того, де він працює:
  1. *Standalone* – автономний – вбудоване кластерне середовище Spark.
  2. *Mesos* – ядро розподілених систем.
  3. *Hadoop YARN*.
  4. *Kubernetes*.
  5. *Amazon EC2*.

Загальна ідея архітектури Spark. Загальний вигляд архітектури Spark представлено на рис. 1.2. Неформально кластер Spark складається з головного вузла («менеджера кластера»), який відповідає за керування програмами Spark, і набору «робочих» (виконавчих) вузлів, які відповідають за виконання завдань, наданих програми Spark (ваших програми, які ви хочете запускати в кластері Spark).

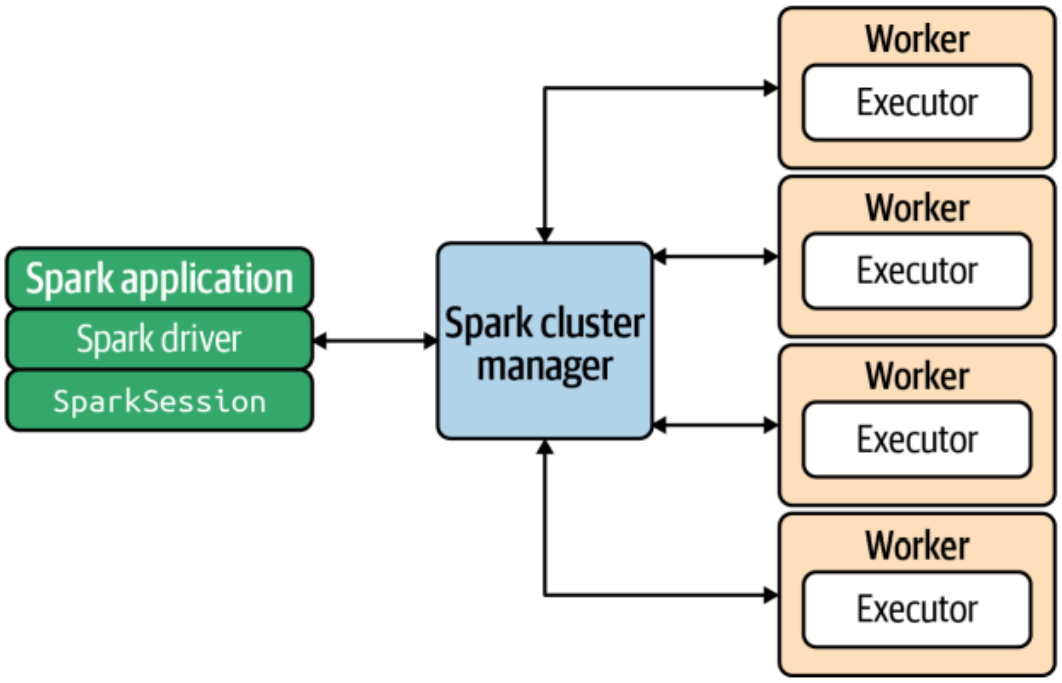

Рис. 1.2. Архітектура Spark.

Залежно від середовища, у якому працює Spark, менеджером кластера, який керуватиме цим кластером серверів, буде автономний менеджер кластера Spark, Kubernetes, Hadoop YARN або Mesos. Коли кластер Spark працює, ви можете надіслати програми Spark менеджеру кластера, який надасть вашій програмі ресурси, щоб ви могли завершити аналіз даних.

Ваш кластер може мати один, десятки, сотні або навіть тисячі робочих вузлів, залежно від потреб вашого бізнесу та вимог вашого проекту. Ви можете запустити Spark на автономному сервері, наприклад MacBook, Linux або Windows, але зазвичай для робочих середовищ Spark запускається на кластері серверів Linux. Щоб запустити програму Spark, вам потрібно мати доступ до кластера Spark і мати програму-драйвер, яка задає необхідні перетворення та дії на RDD даних і надсилає такі запити менеджеру кластера.

Однак, API роботи з даними здебільшого не відрізняється працюєте ви з кластером або з одним вузлом на власному комп'ютері. У цьому курсі всі програми драйверів будуть написані із використанням PySpark та виконуватимуться на одному вузлі.

Які компанії використовують Spark?

- Facebook щодня обробляє 60 ТБ даних. Spark і MapReduce є основою алгоритмів, які використовуються для обробки виробничих даних.
- Viacom зі своїми 170 кабельними, телевізійними та онлайн-мережами приблизно в 160 країнах перетворюється на підприємство, що керується даними, збираючи та аналізуючи петабайти мережевих даних, щоб збільшити лояльність глядачів і дохід.

- Illumina опрацьовує тисячі геномів (це великі дані, які не можуть поміститися на одному сервері або бути обробленими ним) за допомогою Spark, PySpark, MapReduce та розподілених алгоритмів.
- IBM щодня використовує Spark, MapReduce і розподілені алгоритми для масштабування своїх обчислень і операцій.

Spark DataFrame. Spark має два поняття структурованих даних: DataFrame і Dataset. Spark Dataset – це розподілена колекція даних, натомість, Spark DataFrame – це набір даних Spark, організований у стовпці з іменами.

Це означає, що Spark DataFrame є дуже схожою на таблиці, які ми знаємо в реляційних базах даних або в електронних таблицях (як Excel). Отже, у Spark DataFrame кожен стовпець має назву, і всі стовбці мають однакову кількість рядків. Крім того, усі рядки всередині стовпця повинні зберігати дані одного типу, але кожен стовпець може зберігати різні типи даних.

З іншого боку, набори даних Spark вважаються сукупністю будь-яких типів даних. Таким чином, набір даних також може бути набором неструктурованих даних, як-от файли журналу, JSON і XML тощо. Але для більшості програм більш зручно використовувати DataFrame, а не Dataset, для представлення даних.

Основною відмінністю DataFrame від кадрів даних в інших бібліотеках, наприклад Pandas, є розподіленість. Spark DataFrame базується на Spark Dataset. Ці набори даних є колекціями даних, які розподіляються в кластері. Розглянемо дані, представлені в табл. 1.1.

Табл. 1.1. Приклад кадру даних Apache Spark.

ID	**Name**	**Value**
1	Anne	502
2	Carls	432
3	Stoll	444
4	Percy	963
5	Martha	123
6	Sigrid	621

Якщо ви використовуєте Spark у кластері з 4 вузлами (один вузол драйвера, а інші три вузли робочі). Кожен робочий вузол кластера зберігатиме частину цих даних. Таким чином, ви, як програміст, бачитимете, керуватимете та трансформуватимете цю таблицю так, ніби це єдина та об’єднана таблиця. Але під капотом Spark розділить ці дані та збереже їх у вигляді декількох фрагментів у кластері Spark. На рис. 1.3 показано ілюстрацію зберігання цих даних на кластері Spark.

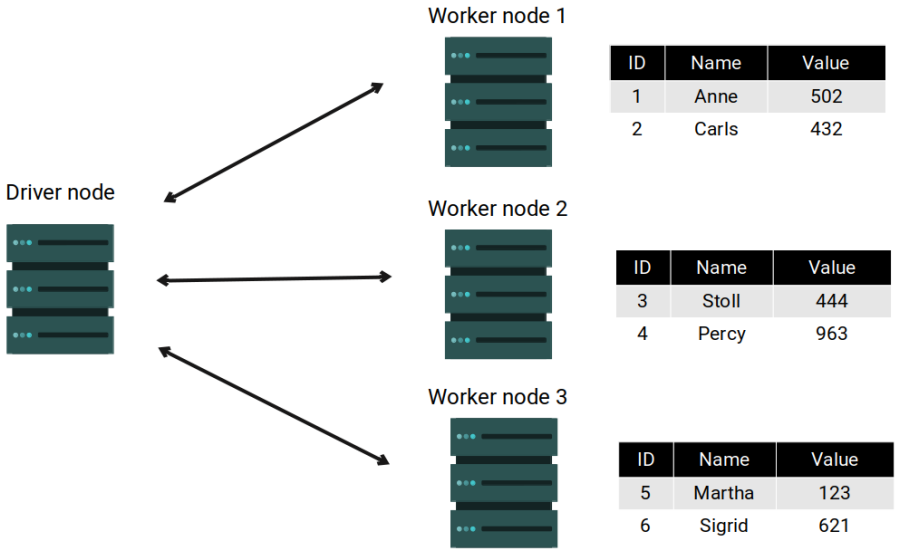

Рис. 1.3. Зберігання єдиного кадру даних на кластері Spark з 4-х вузлів.

Spark DataFrame завжди розбивається на багато дрібних частин, і ці частини завжди розподіляються по кластеру машин. Кожен із цих невеликих фрагментів загальних даних вважається секцією (partition) загального кадру даних.

1.2. Приклад розв'язування задачі

Проведемо налаштування одного вузла Apache Spark на локальному комп'ютері. Для цього необхідно встановити декілька компонентів:

- Python 3.11
- Java Development Kit 17
- Apache Spark 3.5.2
- Apache Hadoop
- PySpark

1. Встановимо Python. Зауважимо, що необхідно встановити саме версію 3.11. Дану версію можна завантажити за посиланням. На початку установки необхідно обов'язково проставити галочку «Add python.exe to PATH», як показано на рис. 1.4.

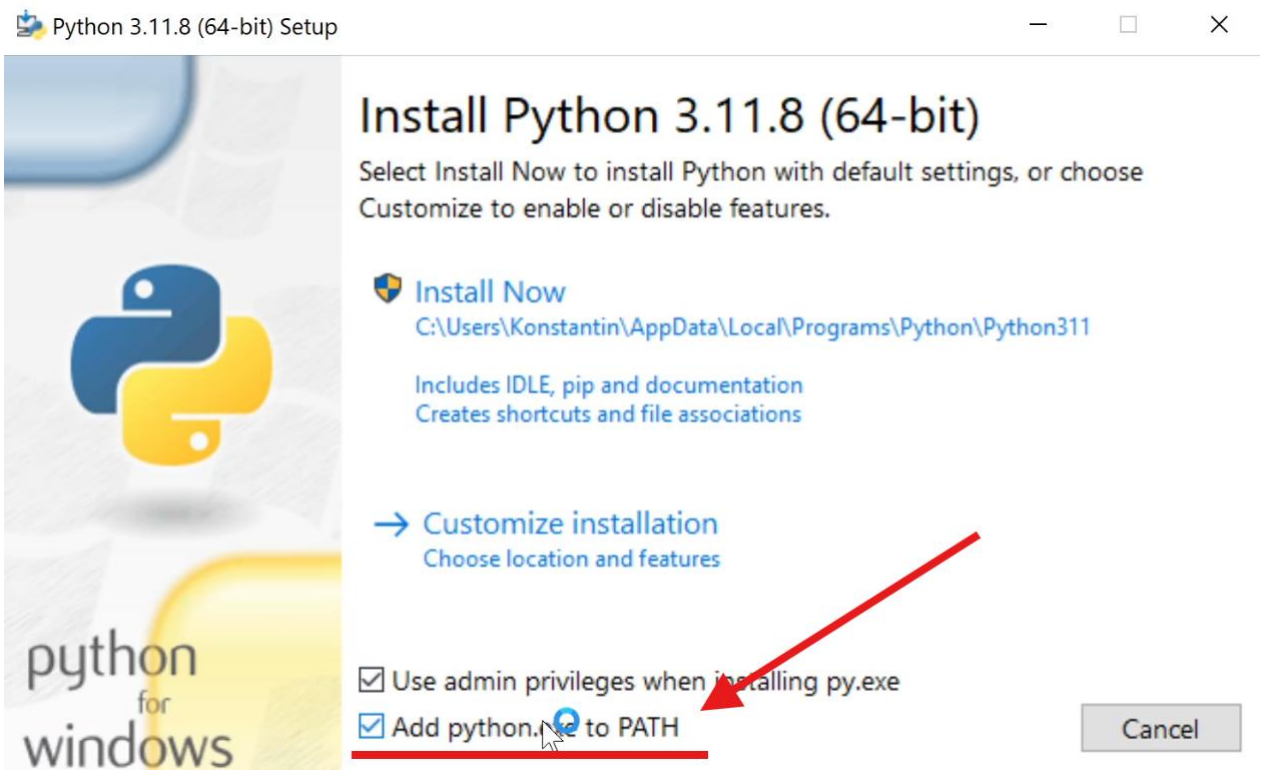

Рис. 1.4. Встановлення Python 3.11.

2. Тепер встановимо Java Development Kit, доступний за [посиланням](). Перед завантаженням необхідно обрати версію JDK 17 та ОС Windows, як показано на рис. 1.5.

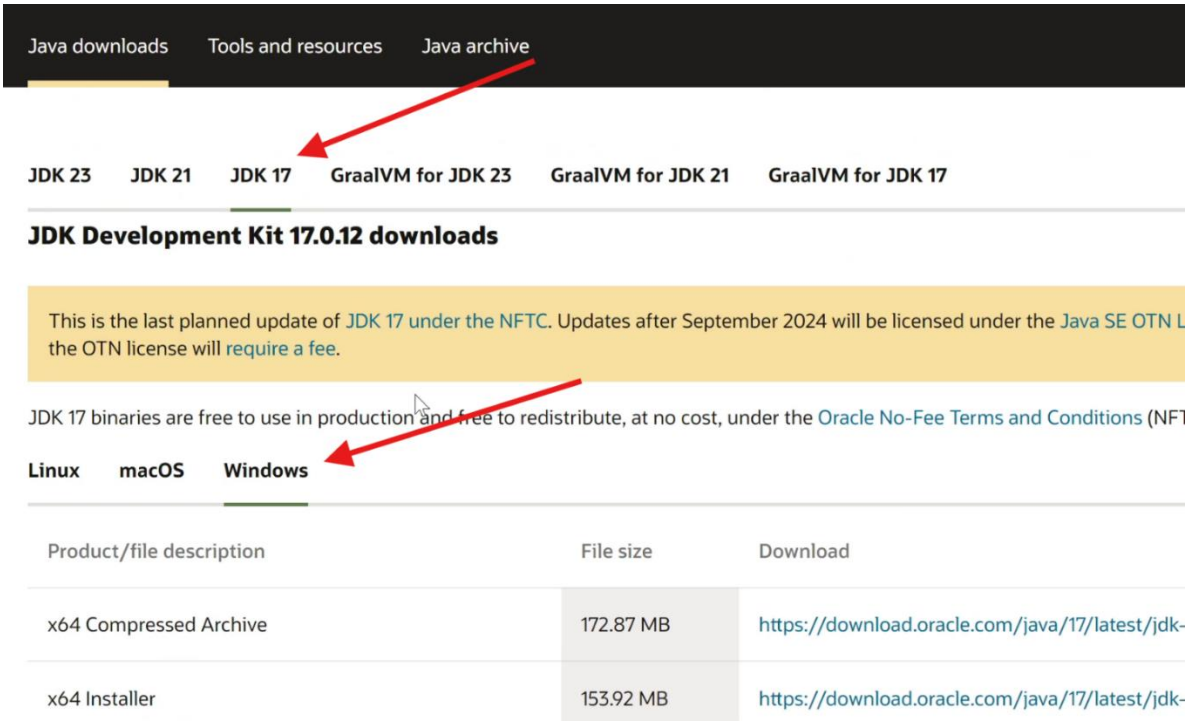

Рис. 1.5. Завантаження Java Development Kit 17.

3. Наступним кроком є завантаження Apache Spark, доступного за посиланням. Завантажуємо версію 3.5.2, як показано на рис. 1.6. Для зручності можна скористатись прямим посиланням на дану версію.

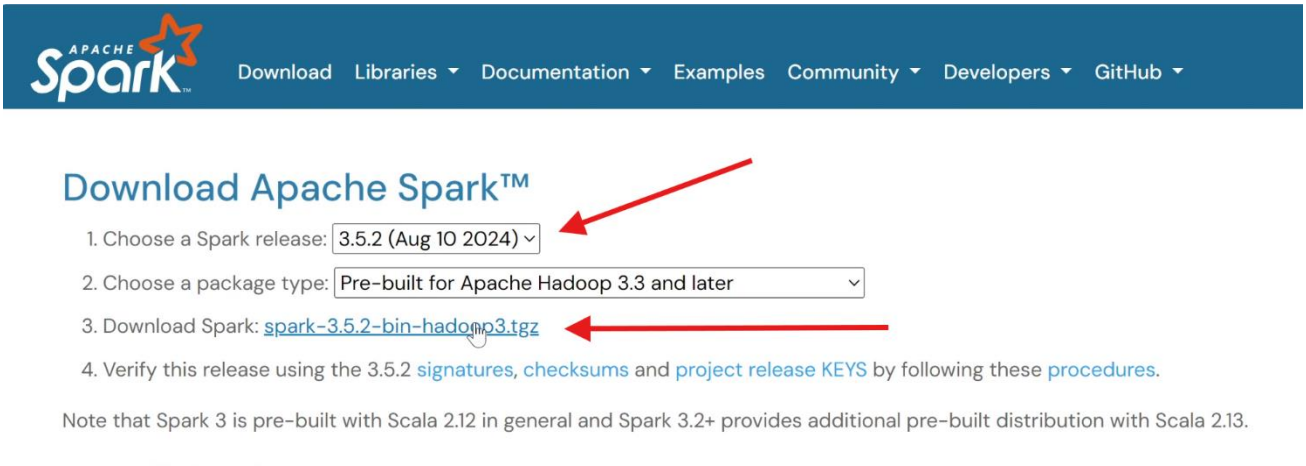

Рис. 1.6. Версія Apache Spark для завантаження.

4. Для встановлення Apache Spark переходимо в корінь диску C:\ та створюємо папку «spark» (рис. 1.7).

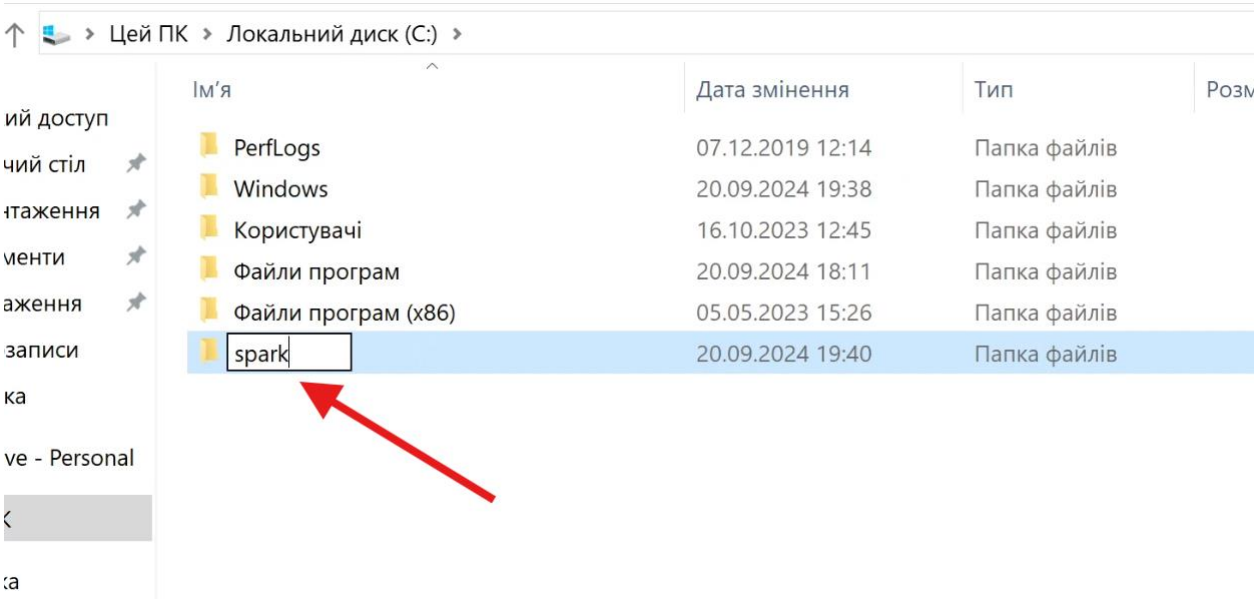

Рис. 1.7. Папка «spark».

5. Копіюємо всі файлі з архіву, завантаженого на кроці 3, в створену папку spark так, щоб зміст папки C:\spark відповідав рис. 1.8.

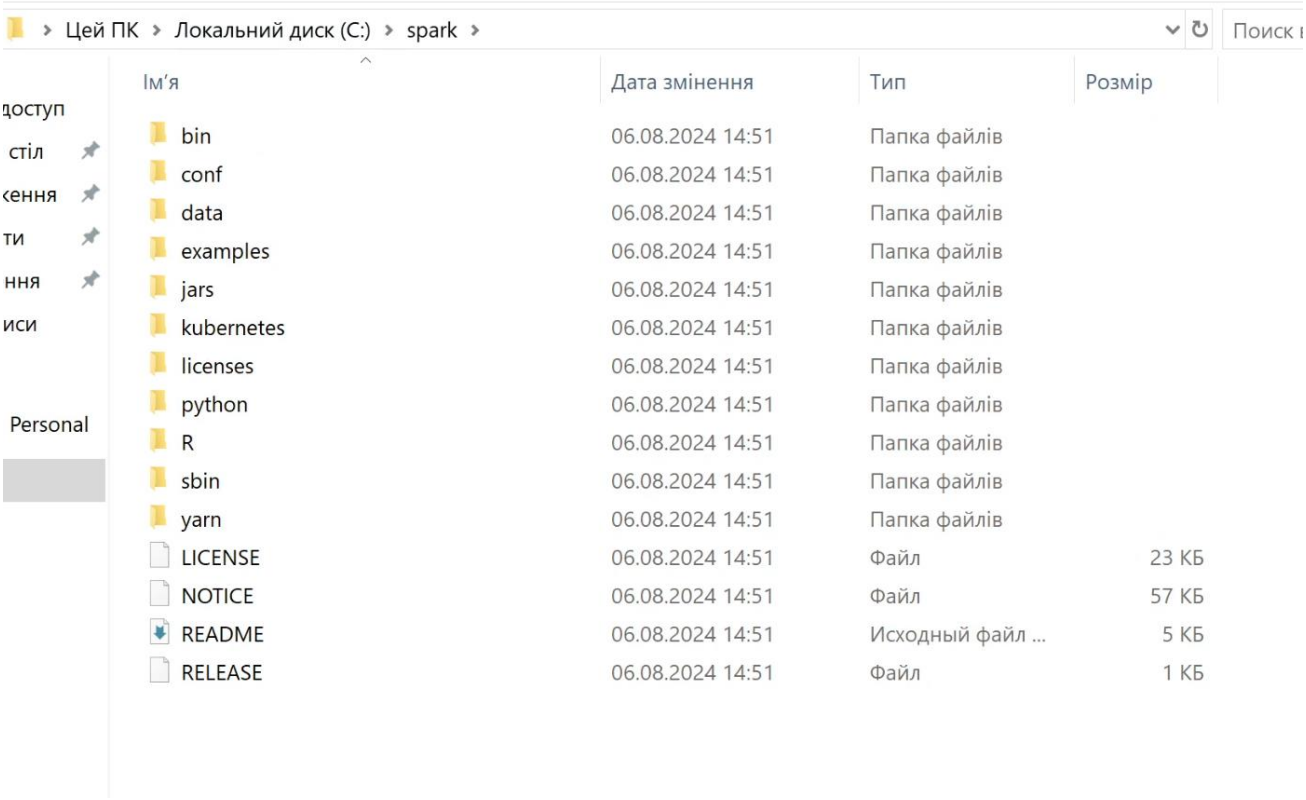

Рис. 1.8. Зміст папки C:\spark

6. Завантажуємо Hadoop 3.3.6. Для цього переходимо за посиланням, натискаємо «…», далі «Download» (рис. 1.9). Завантажився 1 файл winutils.exe.

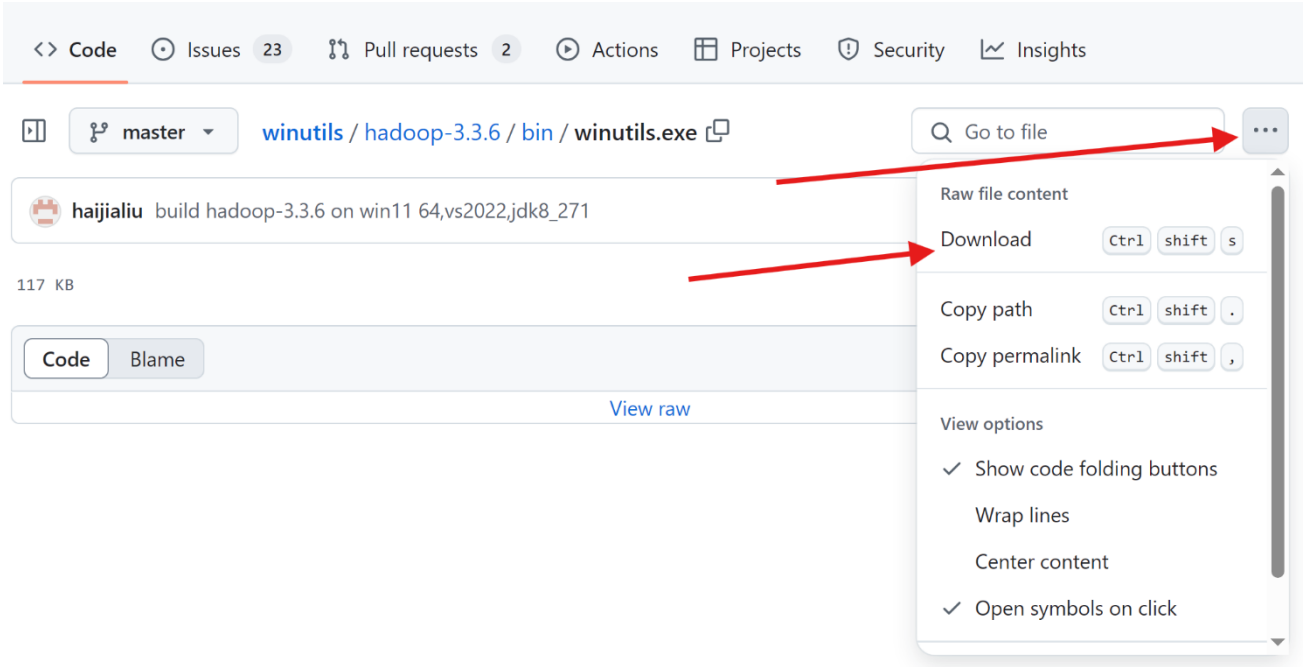

Рис. 1.9. Завантаження Apache Hadoop.

7. Створюємо папку C:\hadoop (рис. 1.10), і в ній папку bin (рис. 1.11).

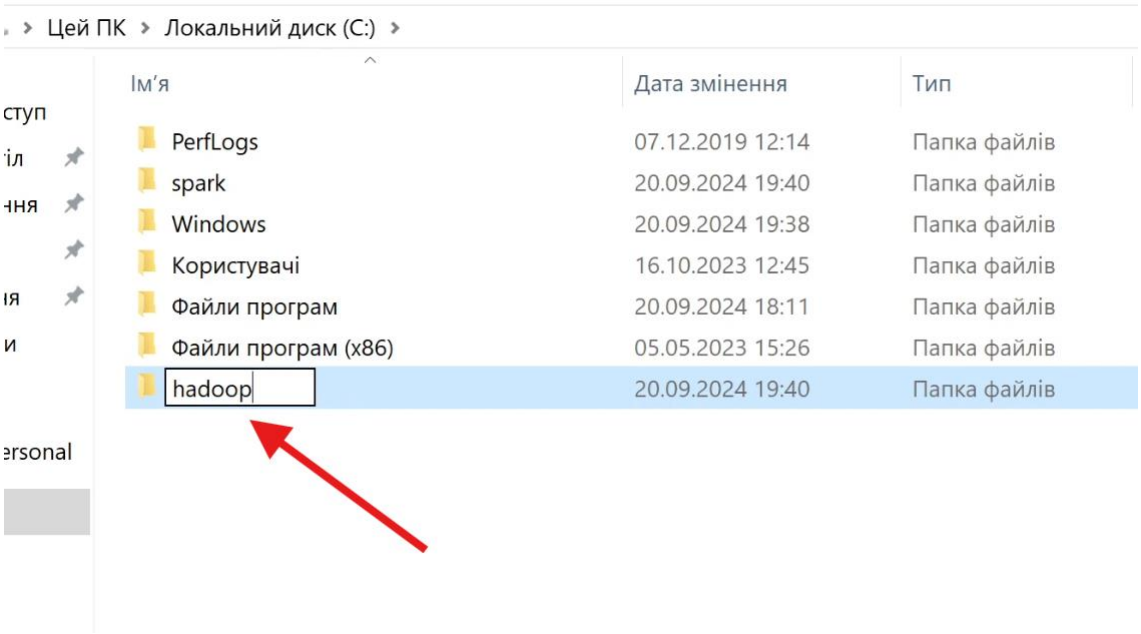

Рис. 1.10. Створення папки hadoop.

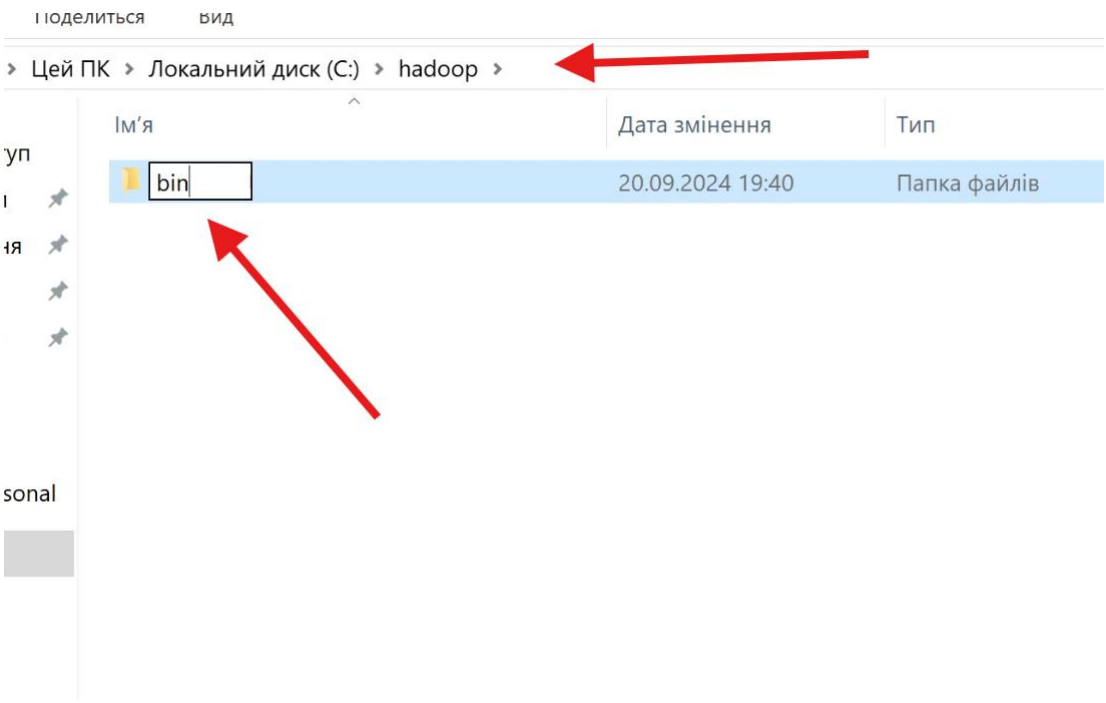

Рис. 1.11. Створення теки bin.

8. Копіюємо файл winutils.exe, завантажений на кроці 6 в папку C:\hadoop\bin (рис. 1.12).

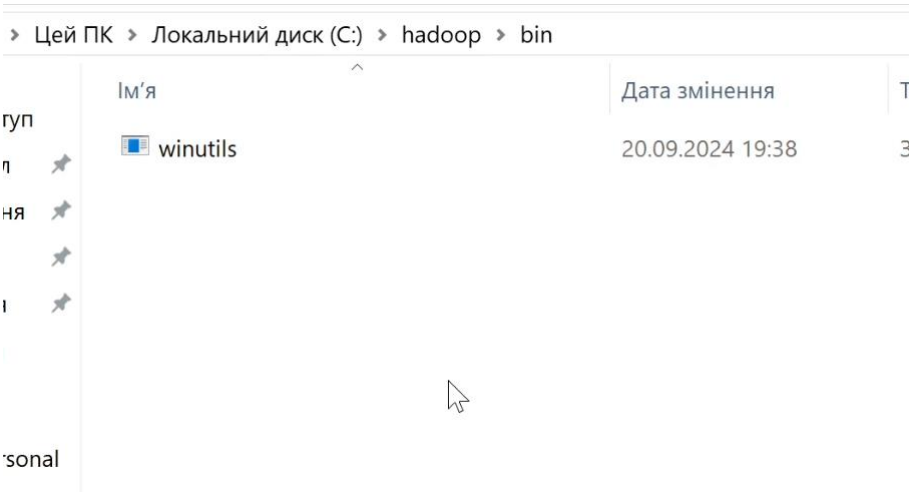

Рис. 1.12. Завершене встановлення Hadoop.

9. Переходимо до створення необхідних змінних оточення системи. Для цього в пошуку вводимо та натискаємо «Редагування змінних оточення системи» (рис. 1.13).

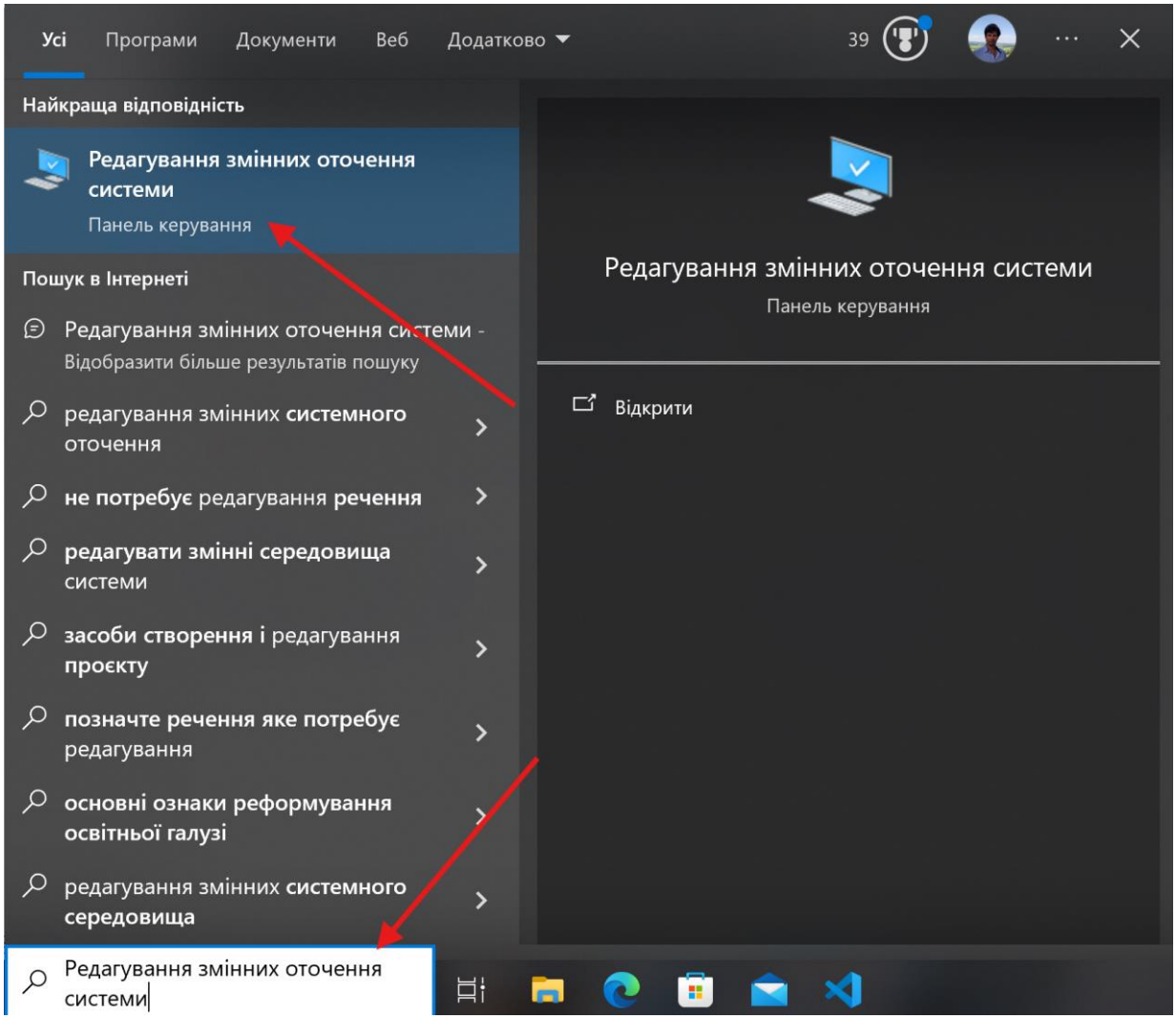

Рис. 1.13. Пошук «Редагування змінних оточення системи».

10. У вікні, що відкрилось, обираємо «Змінні оточення…» (рис. 1.14).

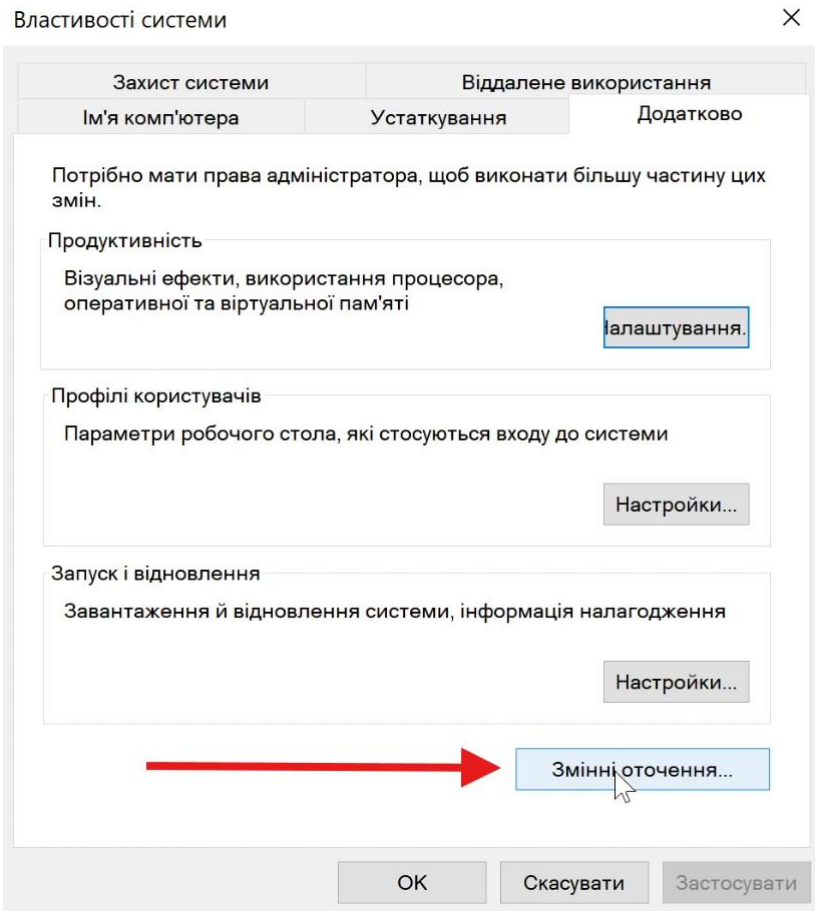

Рис. 1.14. Змінні оточення.

11.Далі створюємо змінні (рис. 1.15):

- Ім’я змінної: JAVA_HOME
  Значення змінної: C:\Program Files\Java\jdk-17
  Як показано на рис. 1.16.
- Ім’я змінної: HADOOP_HOME
  Значення змінної: C:\hadoop
  Як показано на рис. 1.17.
- Ім’я змінної: SPARK_HOME
  Значення змінної: C:\spark
  Як показано на рис. 1.18.
- Ім’я змінної: PYSPARK_PYTHON
  Значення змінної: python
  Як показано на рис. 1.19.

Остаточний результат створення змінних показано на рис. 1.20.

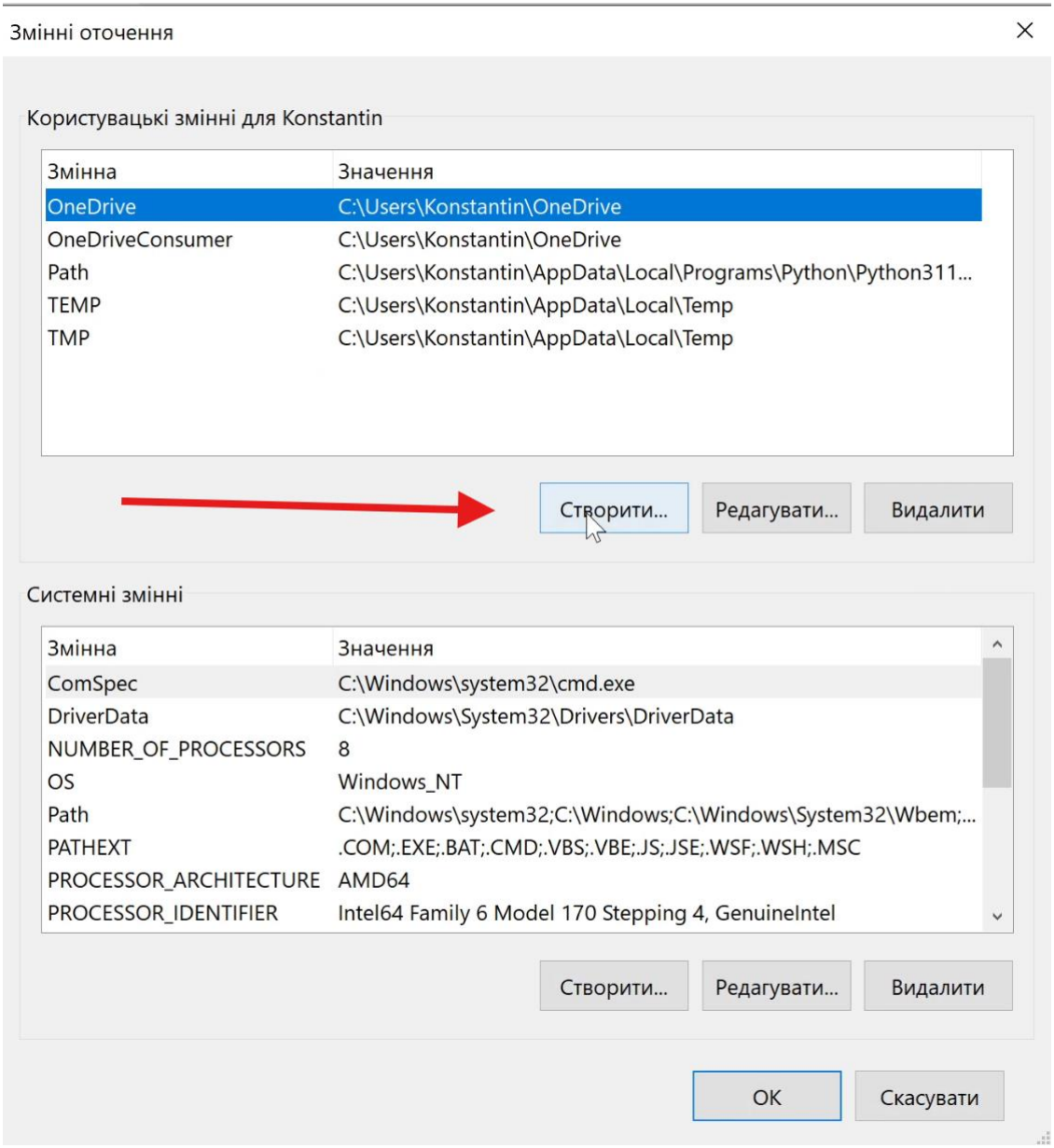

Рис. 1.15.Створення змінних оточення.

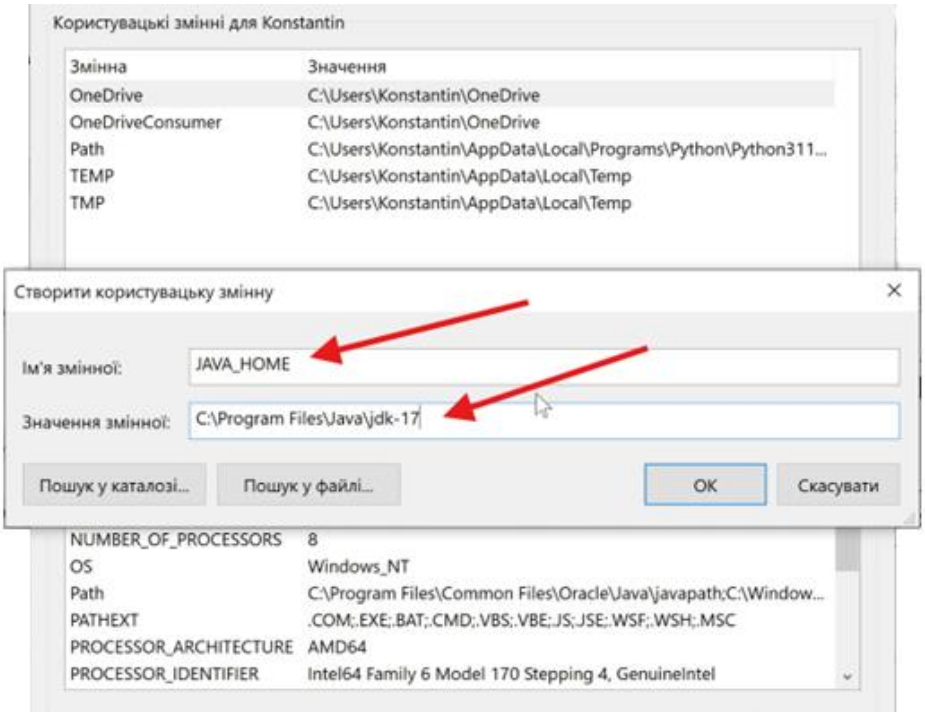

Рис. 1.16. Створення змінної JAVA_HOME.

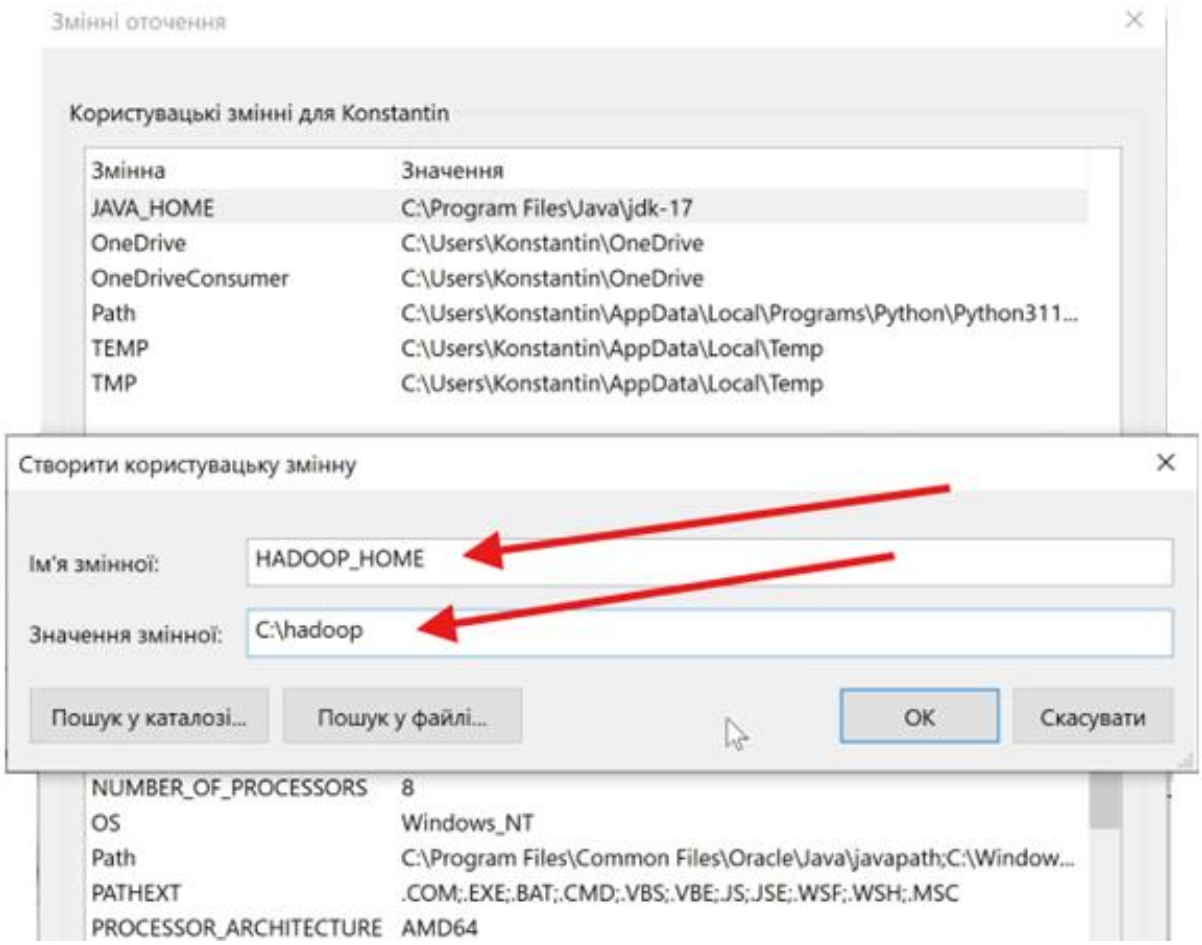

Рис. 1.17. Створення змінної HADOOP_HOME.

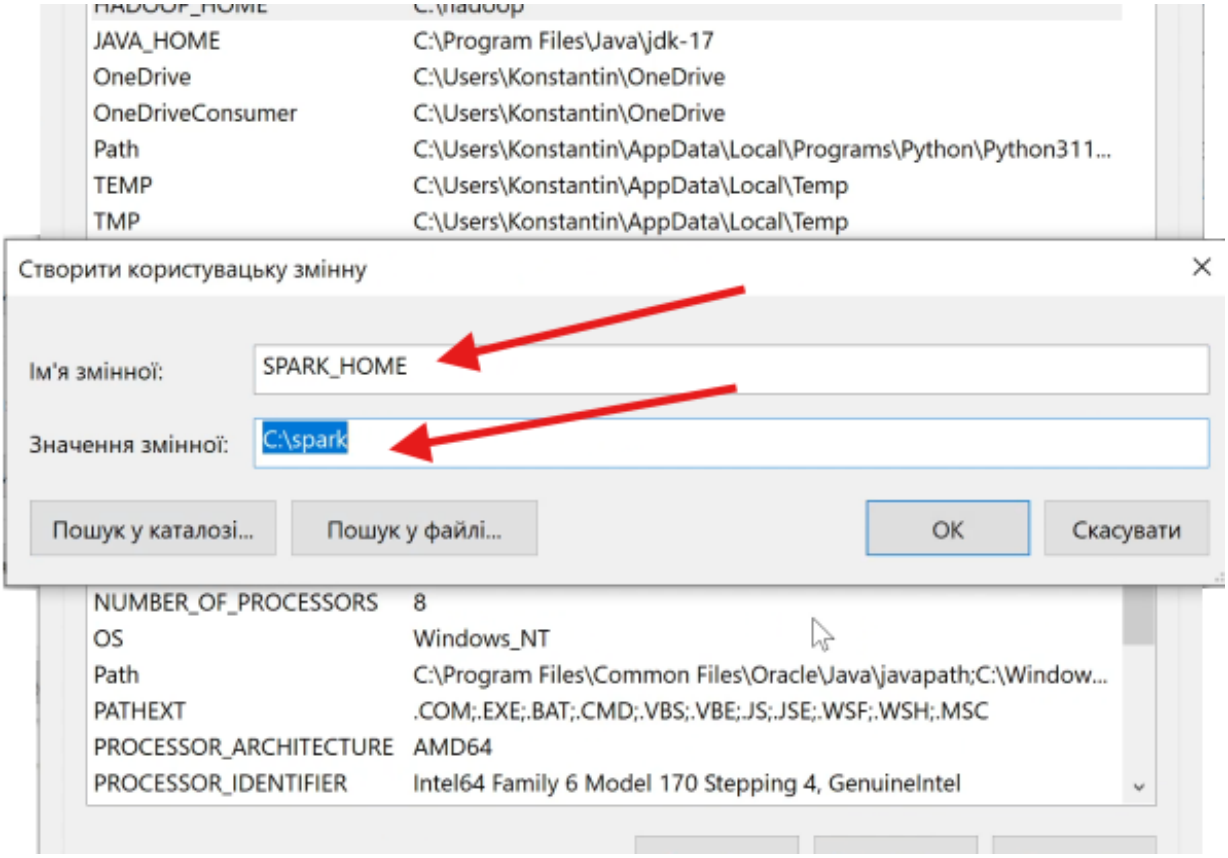

Рис. 1.18. Створення змінної SPARK_HOME.

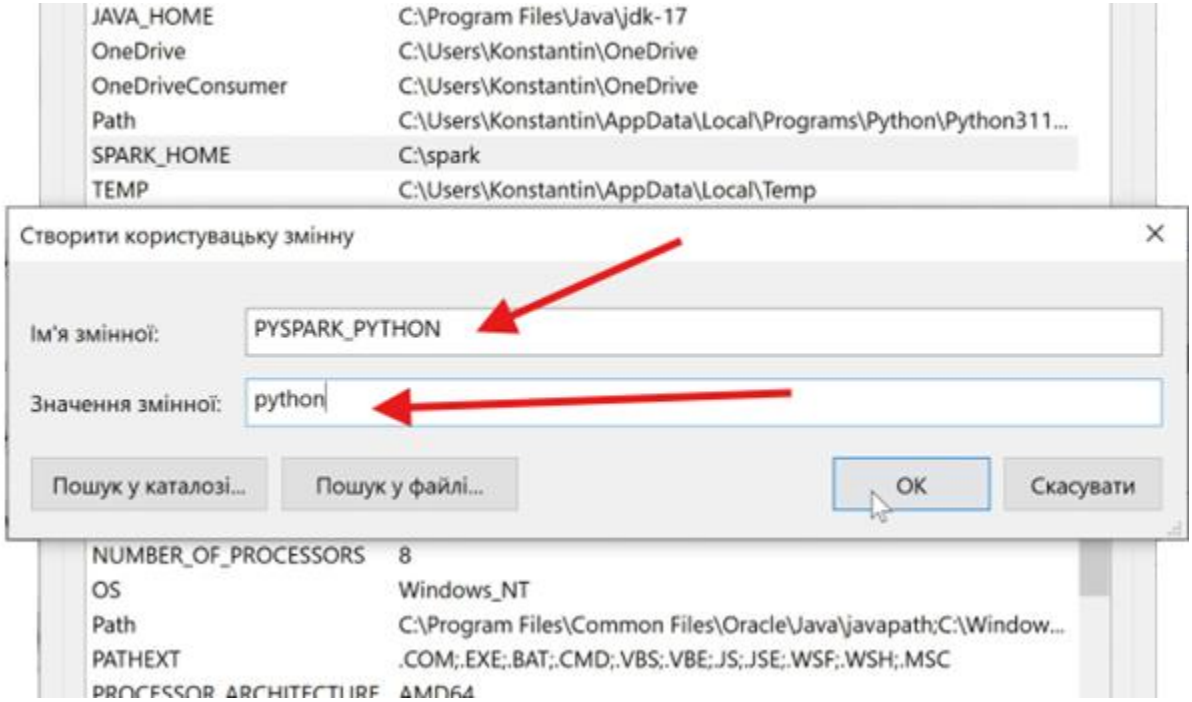

Рис. 1.19. Створення змінної PYSPARK_PYTHON.

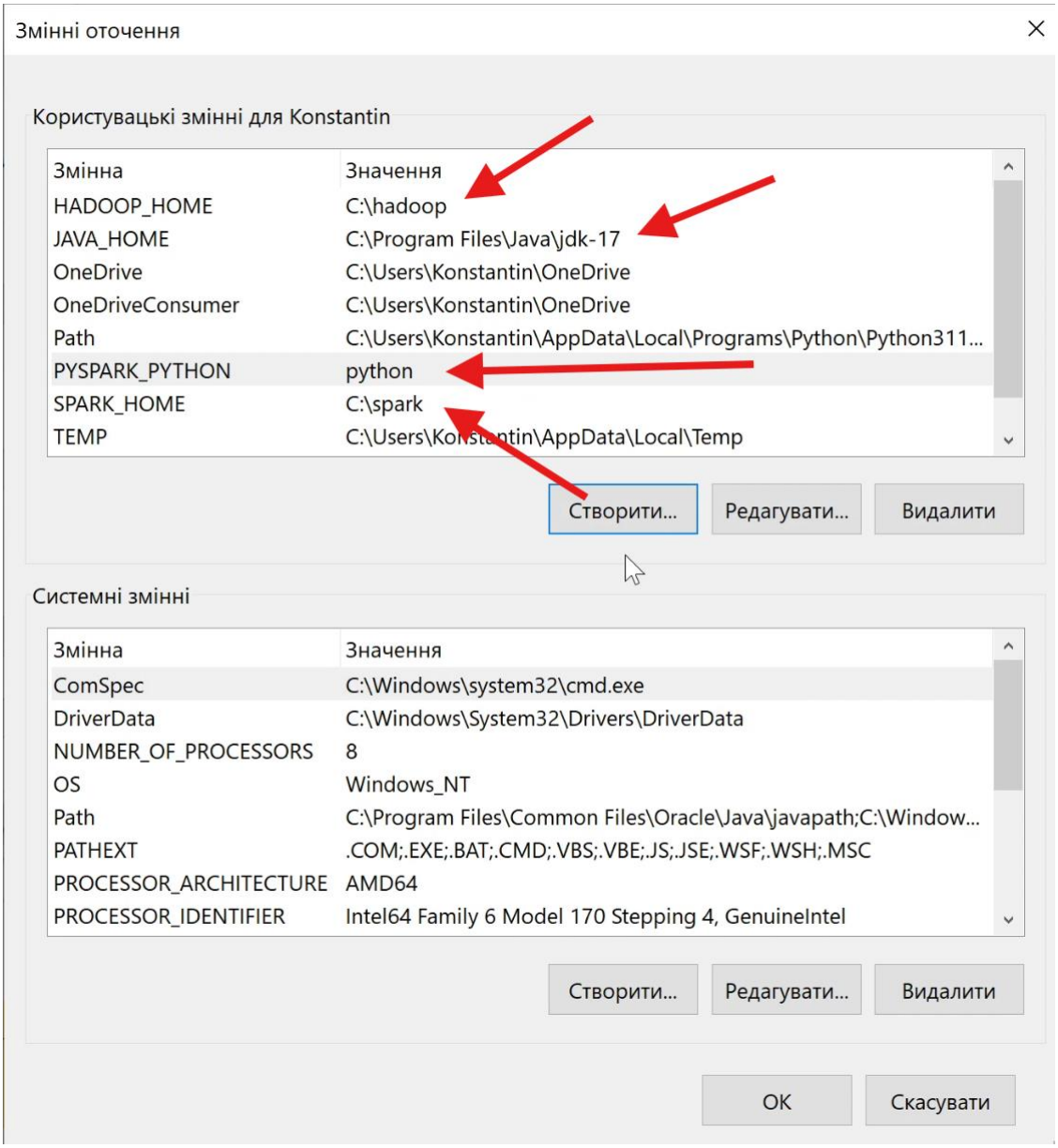

Рис. 1.20. Остаточний результат створення змінних середовища.

12. Редагуємо змінну середовища Path. Для цього обираємо «Path» та натискаємо «Редагувати» (рис. 1.21). У вікні, що відкрилось, натискаємо «Створити» та додаємо «C:\Program Files\Java\jdk-17\bin» (без лапок), аналогічним чином додаємо «C:\spark\bin» та «C:\hadoop\bin». Результат додавання змінних показано рис. 1.22. Для збереження результатів натискаємо «ОК».

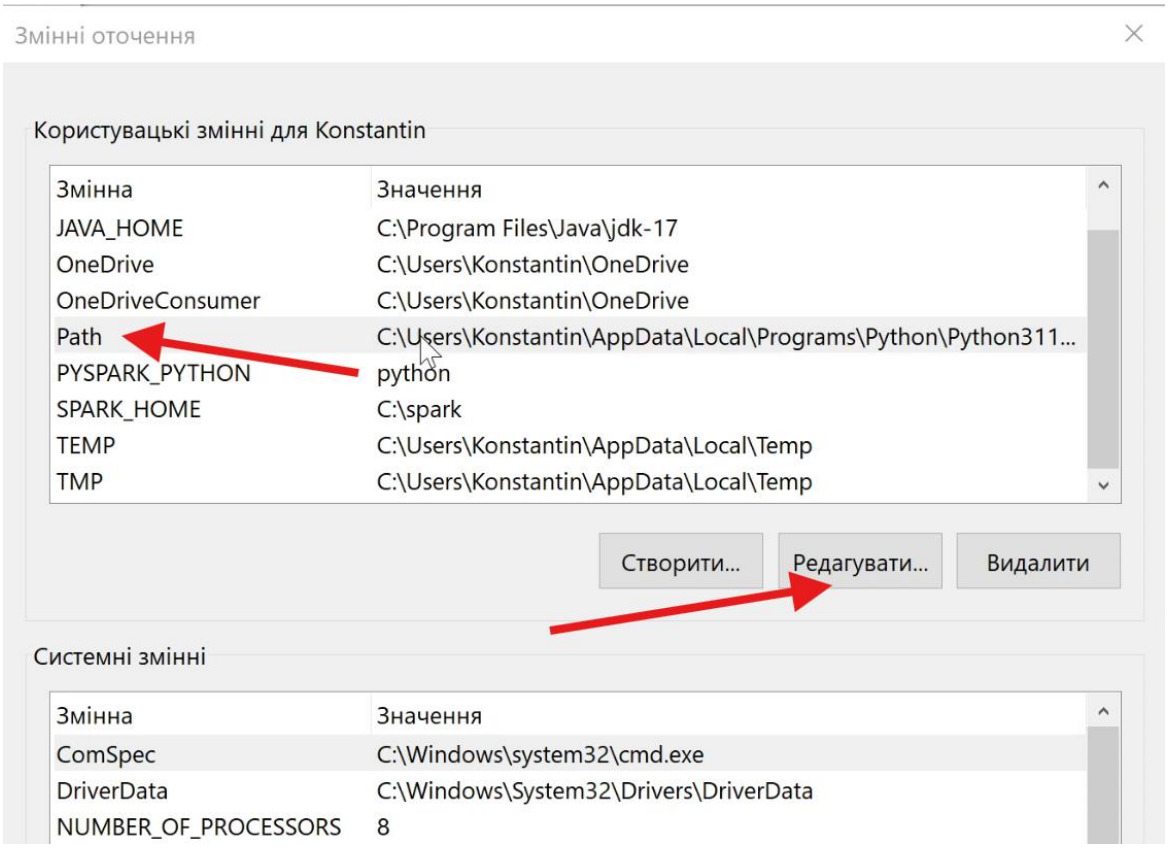

Рис. 1.21. Редагування змінної «Path».

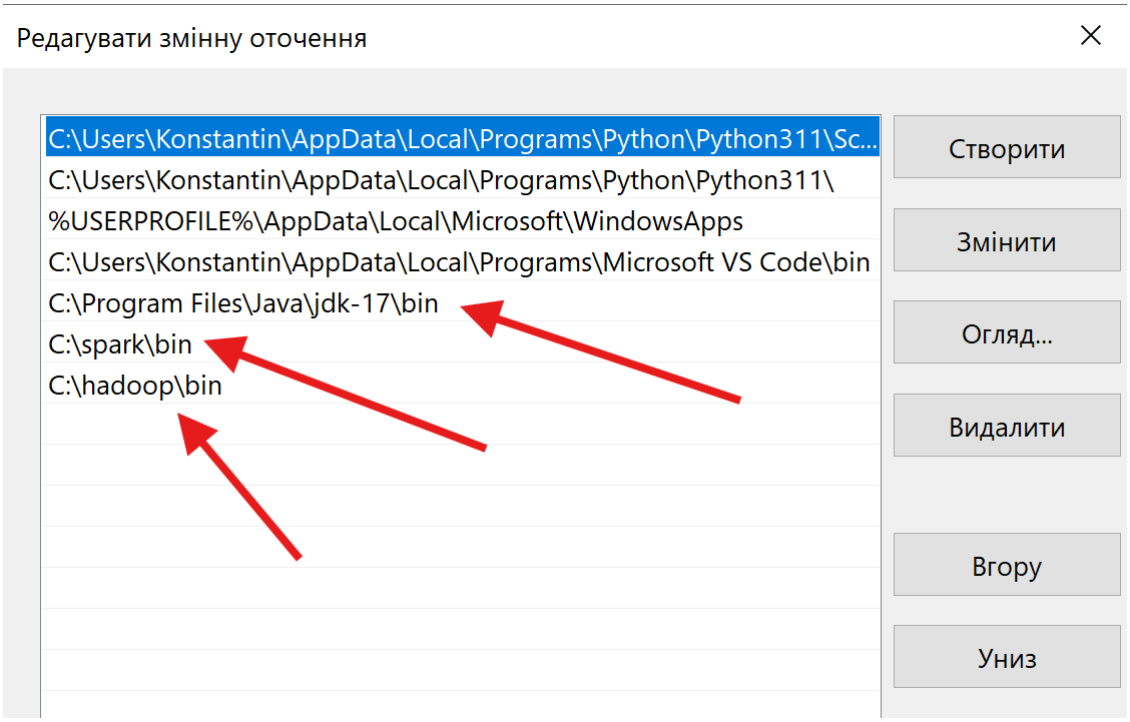

Рис. 1.22. Остаточний результат редагування змінної «Path».

13. Останнім компонентом, що необхідно встановити, є PySpark. Це робиться відкриттям командного рядка та виконанням команди «pip install pyspark» (рис. 1.23). Після цього ви можете запускати Spark програми на власному комп'ютері.

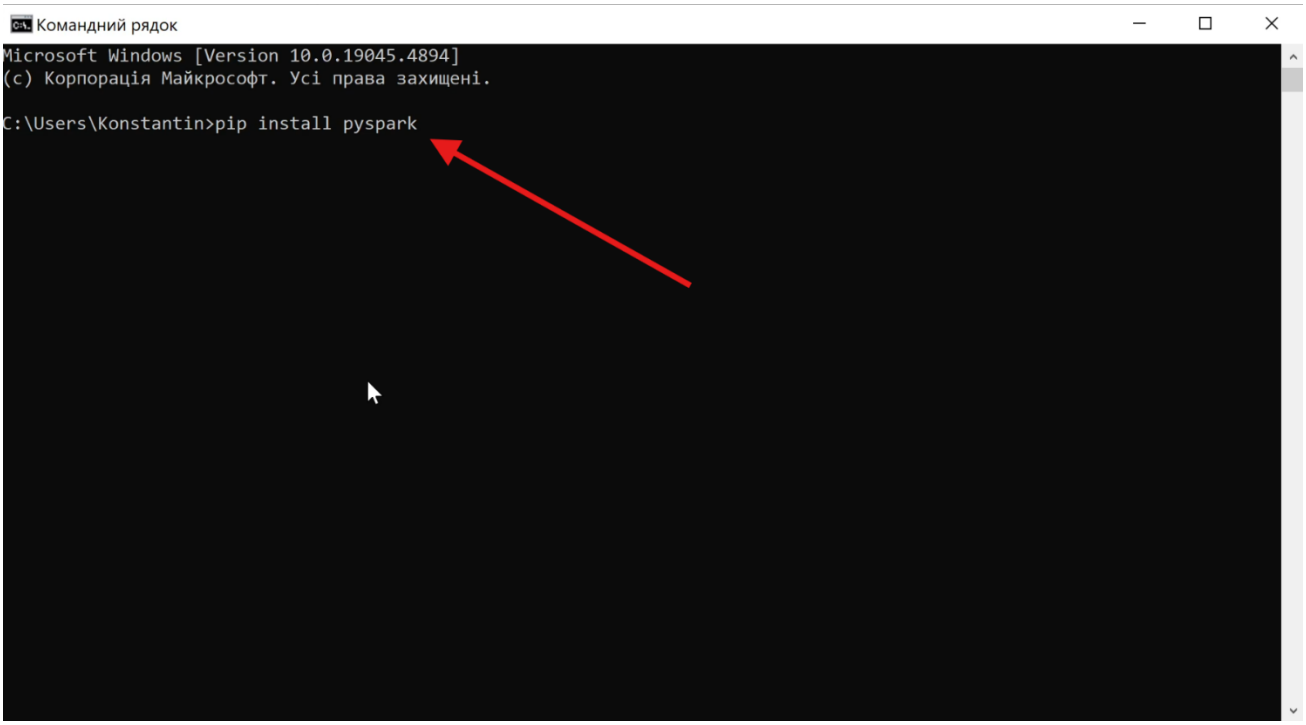

Рис. 1.23. Встановлення PySpark.

Побудова Spark DataFrame. Існує кілька різних методів створення Spark DataFrame. Наприклад, оскільки DataFrame – це в основному набір даних із рядків, ми можемо створити DataFrame із колекції рядків за допомогою методу createDataFrame() із сеансу Spark:

Код

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
from datetime import date
from pyspark.sql import Row

data = [
  Row(id = 1, value = 28.3, date = date(2021,1,1)),
  Row(id = 2, value = 15.8, date = date(2021,1,1)),
  Row(id = 3, value = 20.1, date = date(2021,1,2)),
  Row(id = 4, value = 12.6, date = date(2021,1,3))
]

df = spark.createDataFrame(data)
```

Spark DataFrame у Python є об’єктом класу pyspark.sql.dataframe.DataFrame, як ви бачите нижче:

Код

```
print(type(df))
```

Вивід

```
<class 'pyspark.sql.dataframe.DataFrame'>
```

Якщо ви спробуєте побачити, що знаходиться всередині такого об’єкта, ви отримаєте невеликий опис стовпців, присутніх у DataFrame:

Код

```
print(df)
```

Вивід

```
DataFrame[id: bigint, value: double, date: date]
```

Отже, у наведеному вище прикладі ми використовуємо конструктор Row() (з модуля pyspark.sql) для створення 4 рядків. Метод createDataFrame() поєднує ці 4 рядки разом, щоб сформувати наш новий DataFrame df. Результатом є Spark DataFrame із 4 рядками та 3 стовпцями (ідентифікатор, значення та дата).

Але ви можете використовувати різні методи для створення того самого Spark DataFrame. За допомогою наведеного нижче коду ми створюємо DataFrame під назвою students з двох різних списків Python (даних і стовпців).

Перший список (дані) – це список рядків. Кожен рядок представлено кортежем Python, який містить значення в кожному стовпці. А другий список (стовпці) містить імена для кожного стовпця в DataFrame.

Щоб створити DataFrame студентів, використаємо ці два списки в методі createDataFrame():

Код

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
from datetime import date
from pyspark.sql import Row

data = [
  (12114, 'Anne', 21, 1.56, 8, 9, 10, 9, 'Economics', 'SC'),
  (13007, 'Adrian', 23, 1.82, 6, 6, 8, 7, 'Economics', 'SC'),
  (10045, 'George', 29, 1.77, 10, 9, 10, 7, 'Law', 'SC'),
  (12459, 'Adeline', 26, 1.61, 8, 6, 7, 7, 'Law', 'SC'),
```

```
  (10190, 'Mayla', 22, 1.67, 7, 7, 7, 9, 'Design', 'AR'),
  (11552, 'Daniel', 24, 1.75, 9, 9, 10, 9, 'Design', 'AR')
]

columns = [
  'StudentID', 'Name', 'Age', 'Height', 'Score1',
  'Score2', 'Score3', 'Score4', 'Course', 'Department'
]

students = spark.createDataFrame(data, columns)
print(students)
```

Вивід

```
DataFrame[StudentID: bigint, Name: string, Age: bigint, Height: double,
Score1: bigint, Score2: bigint, Score3: bigint, Score4: bigint,
Course: string, Department: string]
```

Примітка. Насправді Spark виводить деяку системну інформації під час роботи. В прикладах виводу її не наведено. Читач, запускаючи код на своєму комп'ютері, також може ігнорувати її.

Ключовим аспектом Spark є ліниве виконання. Іншими словами, для більшості операцій Spark лише перевірятиме, чи правильний ваш код і чи має він сенс. Spark фактично не запускатиме та не виконуватиме операції, які ви описуєте у своєму коді, якщо ви явно не попросите це за допомогою операції тригера.

Ви можете помітити цю «лінь» у вихідних даних нижче:

Код

```
print(students)
```

Вивід

```
DataFrame[StudentID: bigint, Name: string, Age: bigint, Height: double,
Score1: bigint, Score2: bigint, Score3: bigint, Score4: bigint, Course:
string, Department: string]
```

Зверніть увагу що, коли ми друкуємо об'єкт, який зберігає Spark DataFrame (наприклад, df і students), Spark обчислить і надрукує лише загальну інформацію про структуру вашого Spark DataFrame, а не сам DataFrame.

Отже, як ми можемо побачити наш кадр даних? Як ми можемо візуалізувати рядки та значення, які зберігаються в ньому? Для цього використаємо метод show(). За допомогою якого Spark надрукує таблицю в текстовому форматі, як видно нижче:

Код

```
students.show()
```

Вивід

```
+---------+-------+---+------+------+------+------+------+---------+----------+
|StudentID|   Name|Age|Height|Score1|Score2|Score3|Score4|   Course|Department|
+---------+-------+---+------+------+------+------+------+---------+----------+
|    12114|   Anne| 21|  1.56|     8|     9|    10|     9|Economics|        SC|
|    13007| Adrian| 23|  1.82|     6|     6|     8|     7|Economics|        SC|
|    10045| George| 29|  1.77|    10|     9|    10|     7|      Law|        SC|
|    12459|Adeline| 26|  1.61|     8|     6|     7|     7|      Law|        SC|
|    10190|  Mayla| 22|  1.67|     7|     7|     7|     9|   Design|        AR|
|    11552| Daniel| 24|  1.75|     9|     9|    10|     9|   Design|        AR|
+---------+-------+---+------+------+------+------+------+---------+----------+
```

За замовчуванням цей метод показує лише верхні рядки вашого DataFrame, але ви можете вказати, скільки саме рядків ви хочете бачити, використовуючи show(n), де n – це кількість рядків. Наприклад, можна візуалізувати лише перші 2 рядки df таким чином:

Код

```
students.show(2)
```

Вивід

```
+---------+------+---+------+------+------+------+------+---------+----------+
|StudentID|  Name|Age|Height|Score1|Score2|Score3|Score4|   Course|Department|
+---------+------+---+------+------+------+------+------+---------+----------+
|    12114|  Anne| 21|  1.56|     8|     9|    10|     9|Economics|        SC|
|    13007|Adrian| 23|  1.82|     6|     6|     8|     7|Economics|        SC|
+---------+------+---+------+------+------+------+------+---------+----------+
only showing top 2 rows
```

Отримання назв стовпців. Якщо вам потрібно, ви можете легко зібрати список Python із назвами стовпців, наявними у вашому DataFrame, так само, як ви це зробили б у pandas DataFrame – за допомогою методу columns вашого DataFrame, наприклад:

Код

```
print(students.columns)
```

Вивід

```
['StudentID', 'Name', 'Age', 'Height', 'Score1', 'Score2', 'Score3', 'Score4',
'Course', 'Department']
```

Отримання кількості рядків. Якщо ви хочете дізнатися кількість рядків у Spark DataFrame, просто скористайтеся методом count() цього DataFrame. У результаті Spark створить цей DataFrame і підрахує кількість наявних у ньому рядків.

Код

```
print(students.count())
```

Вивід

```
6
```

Читання даних з файлу. Найчастіше ви будете не створювати кадри даних з нуля, а вивантажувати їх з іншого джерела, будь-то база даних або файл. Розглянемо читання даних з файлу CSV – текстового файлу, де колонки розділені комами.

Створимо файл data.csv в тій же директорії, що і файл Python, із наступним змістом:

data.csv

```
id,"name","occupation","salary"
1,"John Doe","Software Engineer",80000
2,"Jane Smith","Data Scientist",95000
3,"Bob Brown","DevOps Engineer",70000
4,"Alice Johnson","Product Manager",100000
5,"Mike Davis","Cloud Architect",110000
```

Для читання файлу із Spark, використаємо команду `spark.read.csv('data.csv', header=True, inferSchema=True)`.

- `header=True` повідомляє PySpark, що перший рядок у CSV містить назви стовпців.
- `inferSchema=True` дозволяє PySpark автоматично визначати типи даних стовпців на основі їх вмісту.

Код

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Load CSV").getOrCreate()

df = spark.read.csv('data.csv', header=True, inferSchema=True)

df.show()
```

Вивід

```
+---+-------------+-----------------+------+
| id|         name|       occupation|salary|
+---+-------------+-----------------+------+
|  1|     John Doe|Software Engineer| 80000|
|  2|   Jane Smith|   Data Scientist| 95000|
|  3|    Bob Brown|  DevOps Engineer| 70000|
|  4|Alice Johnson|  Product Manager|100000|
|  5|   Mike Davis|  Cloud Architect|110000|
+---+-------------+-----------------+------+
```

Операції над кадрами даних PySpark. Кадри даних PySpark надають різні операції для обробки та аналізу даних. Розглянемо на прикладах деякі з найбільш часто використовуваних операцій.

1. Фільтрування даних

Фільтрування дозволяє вибирати рядки, які відповідають певній умові.

Код

```
from pyspark.sql import SparkSession

# Створюємо простий кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25), ("Bob", 30), ("Charlie", 35)]
df = spark.createDataFrame(data).toDF("name", "age")

# Фільтруємо рядки із віком більше 30
filtered_df = df.filter(df.age > 30)
filtered_df.show()
```

Вивід

```
+-------+---+
|   name|age|
+-------+---+
|Charlie| 35|
+-------+---+
```

2. Групування та агрегація

Групування дозволяє групувати рядки за одним або декількома стовпцями, тоді як агрегування дозволяє виконувати обчислення для кожної групи.

Код

```
from pyspark.sql import SparkSession

# Створюємо кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25000), ("Bob", 30000), ("Charlie", 35000), ("Alice", 14000)]
df = spark.createDataFrame(data).toDF("name", "salary")

# Групуємо за колонкою 'name' та рахуємо середнє за колонкою 'salary'
grouped_df = df.groupBy("name").mean()
grouped_df.show()
```

Вивід

```
+-------+-----------+
|   name|avg(salary)|
+-------+-----------+
|  Alice|    19500.0|
|    Bob|    30000.0|
|Charlie|    35000.0|
+-------+-----------+
```

3. Об’єднання

Об’єднання дозволяє об’єднати два кадра даних на основі спільного стовпця, функція є подібною до знайомої JOIN з SQL.

Код

```
from pyspark.sql import SparkSession

# Створюємо кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25000), ("Bob", 30000), ("Charlie", 35000), ("Alice", 14000)]
df = spark.createDataFrame(data).toDF("name", "salary")

# Створюємо ще одним кадр даних з колонками 'name' та 'occupation'
data2 = [("Alice", "Engineer"), ("Bob", "Doctor"), ("Charlie", "Teacher")]
columns2 = ["name", "occupation"]
df2 = spark.createDataFrame(data2).toDF("name", "occupation")

# Об’єднуємо df та df2 за колонкою 'name'
joined_df = df.join(df2, "name")
joined_df.show()
```

Вивід

```
+-------+------+----------+
|   name|salary|occupation|
+-------+------+----------+
|  Alice| 25000|  Engineer|
```

```
|  Alice| 14000|  Engineer|
|    Bob| 30000|    Doctor|
|Charlie| 35000|   Teacher|
+-------+------+----------+
```

4. Сортування та впорядкування

Сортування дозволяє впорядковувати рядки за одним або кількома стовпцями.

Код

```
from pyspark.sql import SparkSession

# Створюємо кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25000), ("Bob", 30000), ("Charlie", 35000), ("Alice", 14000)]
df = spark.createDataFrame(data).toDF("name", "salary")

# Сортуємо за спаданням колонки 'salary'
sorted_df = df.sort(df.salary.desc())
sorted_df.show()
```

Вивід

```
+-------+------+
|   name|salary|
+-------+------+
|Charlie| 35000|
|    Bob| 30000|
|  Alice| 25000|
|  Alice| 14000|
+-------+------+
```

1.3. Індивідуальне завдання

Завантажити дані для власного варіанту за посиланням на порталі дистанційної освіти.

Виконати наступні запити, використовуючи PySpark. Кожен запит має супроводжуватись скріншотом середовища, кодом та виводом.

Табл. 1.2. Варіанти індивідуальних завдань.

№	Текст завдання
1	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають домашню тварину. 3. Вивести 9 відгуків з id покупки > 500. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
2	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони жіночої статі. 3. Вивести 9 відгуків з id покупки > 500. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони відмінили покупки.
3	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони жіночої статі. 3. Вивести 15 відгуків з оцінкою 2. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
4	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 2 відгуків з id покупки < 2000. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
5	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають дітей. 3. Вивести 5 відгуків з id покупки > 100. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
6	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців із іменем Adams. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
7	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці.

	2. Порахувати кількість покупців про яких відомо, що вони мають дітей. 3. Вивести 2 відгуків з id покупки < 2000. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
8	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають дітей. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 25.
9	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають домашню тварину. 3. Вивести 9 відгуків з id покупки > 500. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони відмінили покупки.
10	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
11	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають домашню тварину. 3. Вивести 3 відгуки з оцінкою 5. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 15.
12	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають дітей. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
13	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 10 відгуків з оцінкою 3. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони відмінили покупки.
14	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони жіночої статі. 3. Вивести 2 відгуків з id покупки < 2000. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
15	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки.

	3. Вивести 3 відгуки з оцінкою 5. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
16	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають дітей. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
17	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають дітей. 3. Вивести 9 відгуків з id покупки > 500. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони відмінили покупки.
18	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 15 відгуків з оцінкою 2. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
19	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 10 відгуків з оцінкою 3. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони відмінили покупки.
20	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців із іменем Adams. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 15.
21	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців із іменем Adams. 3. Вивести 9 відгуків з id покупки > 500. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
22	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 5 відгуків з id покупки > 100. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.

23	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців із іменем Adams. 3. Вивести 3 відгуки з оцінкою 5. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 3 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
24	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони жіночої статі. 3. Вивести 15 відгуків з оцінкою 2. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
25	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають домашню тварину. 3. Вивести 15 відгуків з оцінкою 2. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 15 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 25.
26	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців із іменем Adams. 3. Вивести 2 відгуків з id покупки < 2000. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони не відмінили покупки.
27	1. Прочитати дані індивідуального варіанту та вивести перші 10 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони не мають автівки. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 5 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.
28	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони мають домашню тварину. 3. Вивести 7 відгуків з id покупки < 56. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 15.
29	1. Прочитати дані індивідуального варіанту та вивести перші 5 рядків кожної таблиці. 2. Порахувати кількість покупців із іменем Adams. 3. Вивести 15 відгуків з оцінкою 2. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена тих покупців (не більше 20) про яких відомо, що вони відмінили покупки.
30	1. Прочитати дані індивідуального варіанту та вивести перші 3 рядки кожної таблиці. 2. Порахувати кількість покупців про яких відомо, що вони жіночої статі. 3. Вивести 5 відгуків з id покупки > 100. 4. Знайти середню вартість покупок за кожним customer_id, показати перші 7 рядків. 5. Знайти імена покупців (не більше 20) із вартістю покупки < 10.

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, по кожному запиту: скріншот середовища, код та вивід запиту.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

1.4. Контрольні питання

1. До яких мов програмування Apache Spark має інтерфейс?
2. Для чого потрібні сесія та контекст Spark?
3. Які відповідальності компонент робітника та менеджера кластера?
4. Що таке кадр даних Spark (Spark DataFrame)? Які дані він зберігає? В чому відмінність від Spark Dataset?
5. Яка ключова особливість виконання операцій з кадрами даних Spark?
6. Як відбувається збереження та обробка даних на кластері Spark?
7. Які інструменти необхідно встановити для використання інтерфейсу Spark до мови програмування Python?
8. Як візуалізувати (вивести на консоль) перші рядки таблиці?
9. Які методи необхідно використати для фільтрації даних? Для групування?
10. Як в PySpark можна завантажити дані з файлу?

Практична робота №2 – Прогнозування на великих даних

Мета роботи: закріплення навичок застосування різних методів регресії та класифікації на великих даних, підгонки моделей, прогнозування невідомих значень.

Практична робота присвячена ознайомленню із інструментами розподіленого машинного навчання в PySpark. В ході виконання практичної роботи здобувач закріпить навички щодо підготовки навчальної та тестової вибірок, вирішенню задач регресії та класифікації із використанням інтерфейсу Apache Spark до мови програмування Python PySpark. Для розв'язання задачі практичної роботи за узгодженням з викладачем студент може запропонувати свій набір даних.

2.1. Теоретичні відомості

Класифікація і регресія – найстаріші і найбільш добре вивчені види предиктивної аналітики. Більшість алгоритмів, з якими ви, ймовірно, зіткнетеся в аналітичних пакетах і бібліотеках, є методами класифікації або регресії, такі як метод опорних векторів, логістична регресія, нейронні мережі та глибоке навчання. Спільна нитка, що пов'язує регресію і класифікацію, полягає в тому, що обидві вони припускають прогнозування одного (або декількох) значень на підставі одного (або декількох) інших значень. Для цього їм потрібен набір вхідних і вихідних даних, на яких можна вчитися. Їх потрібно забезпечити як запитаннями, так і еталонними відповідями. З цієї причини вони належать до різновидів навчання з учителем.

PySpark MLlib пропонує реалізації низки алгоритмів класифікації та регресії. До них належать дерева рішень, наївний байєсівський метод, логістична регресія і лінійна регресія. Найприкметніше в цих алгоритмах те, що вони допомагають передбачити майбутнє або, принаймні, передбачити те, чого ми ще не знаємо напевно, наприклад, імовірність того, що ви купите автомобіль, ґрунтуючись на вашій поведінці в Інтернеті; чи є електронний лист спамом, з огляду на слова, які він містить; чи на яких ділянках землі буде зібрано найкращий врожай сільськогосподарських культур з огляду на їхнє місцезнаходження та хімічний склад ґрунту.

У цьому розділі ми зосередимося на популярному і гнучкому типі алгоритму як для класифікації (випадковий ліс), так і для регресії (лінійна регресія). Хоча з реалізацією PySpark можна швидко приступити до роботи, буде корисно зрозуміти основи таких алгоритмів.

Регресія

Проста лінійна регресія. Важливим питанням аналізу даних є відповідь на запитання: чи пов'язана змінна X (або, що більш ймовірно, $X_1, \dots, X_p$) зі змінною

Y, і якщо так, то в чому цей зв'язок полягає, і чи можемо ми його використати для того, щоб передбачити Y?

Проста лінійна регресія, або парна лінійна регресія, моделює зв'язок між величиною однієї змінної та величиною другої, наприклад у міру збільшення X збільшується і Y. Або ж у міру збільшення X зменшується і Y. Кореляція – ще один спосіб виміряти те, яким чином дві змінні зв'язані між собою. Різниця між ними полягає в тому, що кореляція вимірює силу зв'язку між двома змінними, тоді як регресія оцінює природу зв'язку кількісно.

Рівняння регресії. Проста лінійна регресія оцінює, наскільки саме зміниться Y коли X змінюється на деяку величину. Для коефіцієнта кореляції змінні X та Y взаємозамінні. У разі регресії ми намагаємося передбачити змінну Y зі змінної X, використовуючи лінійний зв'язок (тобто прямий):

$$Y = b_0 + b_1 X$$

Ця формула читається, як «Y дорівнює b_1, помножене на X плюс константа b_0». Компонент рівняння b_0 називається перетином (або константою), а b_1 – нахилом по відношенню до осі x. Змінна Y називається *відгуком* або *залежною змінною*, оскільки вона залежить від X. Змінна X називається *провісником* (предиктором, від англ. predictor), або *незалежною змінною*. Спільнота машинного навчання тяжіє до використання інших термінів, називаючи Y *метою* та X – *вектором ознак*.

У регресійному аналізі важливими поняттями є *підігнані значення* та *залишки*. Як правило, дані не лягають рівно на пряму, тому рівняння регресії має включати явний залишковий член e_i:

$$Y_i = b_0 + b_1 X_i + e_i.$$

Підігнані значення, також звані передбаченими значеннями, у типовій ситуації позначаються як $\hat{Y}_i$ (Y з капелюхом). Вони задаються такою формулою:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1 X_i.$$

Позначення $\hat{b}_0$ і $\hat{b}_1$ свідчить, що ці коефіцієнти оцінюються відносно відомим, тобто є оціночними.

Найменші квадрати. Яким чином виконується підгонка моделі до даних? Коли існує чіткий зв'язок, ви можете подумки уявити підгонку прямої вручну. Насправді пряма регресії є оцінкою, яка мінімізує значення суми квадратичних залишків, також іменованих *сумою квадратів залишків* чи *залишковою сумою квадратів* (residual sum of squares, RSS):

$$RSS = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2 = \sum_{i=1}^{n} \left(Y_i - \hat{b}_0 - \hat{b}_1 X_i\right)^2$$

Оцінки $\hat{b}_0$ та $\hat{b}_1$ – це значення, які мінімізують суму квадратів залишків (RSS).

Метод мінімізації суми квадратів залишків називається *регресією на основі найменших квадратів*, чи регресією *на основі звичайних найменших квадратів*

(ЗНК). Цей метод часто приписується Карлу Фрідріху Гауссу, німецькому математику, але він був вперше опублікований в 1805 французьким математиком Андре-Марі Лежандром (Adrien-Marie Legendre). Регресію на основі найменших квадратів можна легко та швидко обчислити за допомогою стандартної статистичної обчислювальної системи.

Множинна лінійна регресія. Коли провісників кілька, рівняння регресії просто розширюється їх розміщення:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p + e$$

Замість прямої тепер у нас лінійна модель – зв'язок між кожним коефіцієнтом та його змінною (ознакою) є лінійним.

Всі інші поняття з простої лінійної регресії, такі як підгонка найменшими квадратами, підігнані значення та залишки, відносяться і до умов множинної лінійної регресії. Наприклад, підігнані значення задаються наступною формулою:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1X_{1,i} + \hat{b}_2X_{2,i} + \cdots + \hat{b}_pX_{p,i}$$

Класифікація

Дерева рішень. У цьому параграфі ми розглянемо ще одне сімейство моделей машинного навчання – дерево рішень.

Дерево рішень передбачає значення цільової змінної за допомогою застосування послідовності простих розв'язувальних правил (які називаються предикатами). Цей процес у певному сенсі узгоджується з природним для людини процесом ухвалення рішень.

Хоча узагальнювальна здатність дерев, що вирішують, невисока, їхні передбачення обчислюються доволі просто, через що дерева рішень часто використовують як цеглинки для побудови ансамблів – моделей, що роблять передбачення на основі агрегації передбачень інших моделей. Про них ми поговоримо далі.

Приклад вирішального дерева. Почнемо з невеликого прикладу. На рис. 2.1 зображено дерево, побудоване для задачі класифікації на п'ять класів.

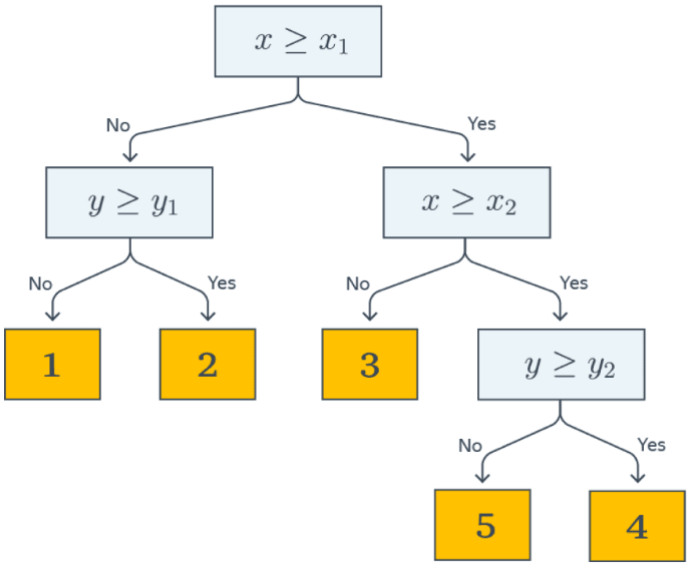

Рис. 2.1. Приклад дерева рішень.

Об'єкти в цьому прикладі мають дві ознаки з дійсними значеннями: X і Y. Рішення про те, до якого класу буде віднесено поточний об'єкт вибірки, ухвалюватиметься за допомогою проходу від кореня дерева до деякого листа.

У кожному вузлі цього дерева міститься предикат. Якщо предикат вірний для поточного прикладу з вибірки, ми переходимо до правого нащадка, якщо ні – до лівого. У цьому прикладі всі предикати – це просто взяття порога за значенням якоїсь ознаки:

$$B(x, j, t) = [x_j \leq t] \tag{2.1}$$

У листках записані передбачення (наприклад, мітки класів). Щойно ми дійшли до листка, ми присвоюємо об'єкту відповідь, записану у вершині.

На рис. 2.2 візуалізовано процес побудови розв'язувальних поверхонь, породжуваних деревом (права частина картинки).

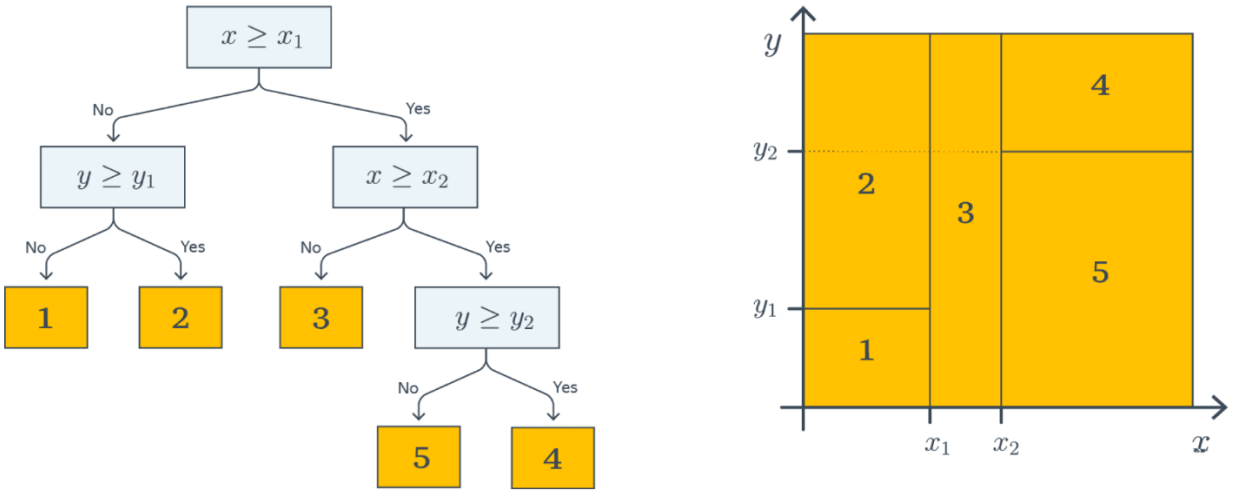

Рис. 2.2. Процес побудови розв’язувальних поверхонь.

Кожен предикат породжує поділ поточної підмножини простору ознак на дві частини. На першому етапі, коли відбувався поділ за $[X \leq X_1]$, всю площину було поділено на дві відповідні частини. На наступному рівні частина площини, для якої виконується $[X \leq X_1]$, була поділена на дві частини за значенням другої ознаки $Y \leq Y_1$ – так утворилися області 1 і 2. Те саме повторюється для правої частини дерева – і так далі до листових вершин: вийде п'ять областей на площині. Тепер будь-якому об'єкту вибірки буде присвоюватися один із п'яти класів залежно від того, в яку з утворених областей він потрапляє. Цей приклад добре демонструє, зокрема, те, що дерево здійснює кусково-постійну апроксимацію цільової залежності.

Випадковий ліс. Передбачення одного дерева є не надто точним. Для підвищення якості окремі дерева рішень можна об’єднати в так званий «випадковий ліс». Для об’єднання достатньо, щоб окремі дерева рішень були певною мірою не схожі один на одного.

Побудуємо ансамбль алгоритмів, де базовий алгоритм – це дерево рішень. Будемо будувати за такою схемою:

1. Для побудови i-го дерева:
    a. Спочатку, як у звичайному беггінгу, з навчальної вибірки X вибирають із поверненням випадкову підвибірку X^i такого самого розміру, що й X.
    b. У процесі навчання кожного дерева в кожній вершині випадково вибирають $n < N$ ознак, де N – повне число ознак (метод випадкових підпросторів), і серед них шукають оптимальне розбиття. Такий прийом якраз дає змогу керувати ступенем скорельованості базових алгоритмів.

2. Щоб отримати передбачення ансамблю на тестовому об'єкті, усереднюємо окремі відповіді дерев (для регресії) або беремо найпопулярніший клас (для класифікації).

3. Ми побудували Random Forest (випадковий ліс) – комбінацію беггінгу і методу випадкових підпросторів над вирішальними деревами.

2.2. Приклад розв’язування задачі

Нехай початкові дані задано файлі products.csv, що надано нижче.

products.csv

```
id,f1,f2,f3,f4,f5,rating,category
1,0.8,0.9,0.7,0.6,0.4,0.85,3
2,0.9,0.5,0.8,0.7,0.3,0.75,1
3,0.7,0.6,0.5,0.4,0.8,0.65,2
4,0.6,0.7,0.9,0.5,0.1,0.55,4
5,0.3,0.4,0.6,0.8,0.9,0.6,5
6,0.5,0.8,0.7,0.9,0.2,0.75,3
7,0.4,0.1,0.3,0.6,0.7,0.45,2
8,0.9,0.6,0.5,0.7,0.4,0.75,4
9,0.1,0.3,0.4,0.8,0.6,0.45,1
10,0.7,0.2,0.5,0.9,0.3,0.55,5
11,0.6,0.4,0.8,0.5,0.1,0.45,2
12,0.8,0.7,0.3,0.6,0.9,0.65,3
13,0.5,0.9,0.4,0.1,0.7,0.55,4
14,0.2,0.8,0.6,0.3,0.5,0.45,1
15,0.9,0.5,0.7,0.4,0.2,0.65,3
16,0.7,0.6,0.8,0.9,0.1,0.75,5
17,0.4,0.3,0.5,0.2,0.6,0.35,2
18,0.6,0.7,0.9,0.8,0.3,0.75,4
19,0.3,0.1,0.4,0.5,0.7,0.45,1
20,0.5,0.2,0.6,0.9,0.8,0.55,3
```

Для побудови та навчання моделі регресії використаємо інструментарії модуля pyspark.ml з бібліотеки PySpark.

Крок 1. Імпорт необхідних залежностей.

Код

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
```

Крок 2. Створення сесії PySpark.

Код

```
spark = SparkSession.builder.appName("LinearRegressionExample").getOrCreate()
```

Крок 3. Читання даних з файлу csv. Файл csv має знаходитись в тій же директорії, що і скрипт Python.

Код

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)
```

Крок 4. Створення вектору ознак (незалежних змінних) використовуючи клас `VectorAssembler`.

Код

```
assembler = VectorAssembler(inputCols=["f1", "f2", "f3", "f4", "f5"],
outputCol="features")
assembled_df = assembler.transform(df)
```

Крок 5. Розбиття даних на тренувальну та тестову частину використовуючи функцію `randomSplit`. В прикладі нижче відкладаємо 80% даних для тренування і 20% для тестування моделі.

Код

```
train_df, test_df = assembled_df.randomSplit([0.8, 0.2])
```

Крок 6. Будуємо модель лінійної регресії та вказуємо, що вхідні ознаки збережені в `features`, а вихідні в `rating`.

Код

```
lr = LinearRegression(labelCol="rating", featuresCol="features")
```

Крок 7. Здійснюємо навчання моделі регресії.

Код

```
model = lr.fit(train_df)
```

Крок 8. Здійснюємо передбачення на тестових даних та виводимо перші 5 рядків.

Код

```
predictions = model.transform(test_df)
predictions.show(5)
```

Крок 9. Оцінюємо отримані передбачення як корінь із середньоквадратичної помилки.

Код

```
evaluator = RegressionEvaluator(labelCol="rating", predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(rmse)
```

2.3. Індивідуальне завдання

Після використання моделі машинного навчання з прикладу було вирішено проблему оцінки рейтингу товарів. Однак, задачу визначення категорії (класифікація) товару вирішено так і не було. Цю задачу поклали на вас. Для цього вам необхідно:

1. Завантажити дані для власного варіанту за посиланням на порталі дистанційної освіти.
2. Здійснити розбиття даних так, щоб в тестовій частині було x% даних (табл. 2.1), а ті, що залишились в тренувальній.
3. Для навчання класифікатора використайте `RandomForestClassifier` з пакету `pyspark.ml.classification` та `MulticlassClassificationEvaluator` з пакету `pyspark.ml.evaluation`.
   Побудуйте класифікатор, що визначає категорію товару за ознаками f1,…f5. В `RandomForestClassifier` задати кількість дерев, використовуючи параметр `numTrees` (кількість дерев визначається індивідуальним варіантом з таблиці 2.1), а в `MulticlassClassificationEvaluator` задати `metricName="accuracy"`
4. Проаналізувати важливість кожної з ознак. Знайти таку мінімальну комбінацію ознак, що призводить до падіння якості не більше, ніж 10%.

Табл. 2.1. Варіанти індивідуальних завдань.

№	Прикладів в тестовій вибірці (%)	Кількість дерев (`numTrees`)
1	28	17
2	17	13
3	26	15
4	27	12
5	27	18
6	15	12

№	Прикладів в тестовій вибірці (%)	Кількість дерев (`numTrees`)
7	30	17
8	18	15
9	20	19
10	20	16
11	28	8
12	21	10
13	16	20
14	23	20
15	22	13
16	19	17
17	23	14
18	18	19
19	18	19
20	23	15
21	28	8
22	19	11
23	30	12
24	23	8
25	28	10
26	28	9
27	25	16
28	22	13
29	25	10
30	29	14

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, розрахунки, необхідні описи, вихідний код програм та скріншот середовища.

Назва файлу: Прізвище_група
Роботи, виконані не за варіантом, не приймаються.

2.4. Контрольні питання

1. Випишіть рівняння множинної регресії. Поясніть значення кожного з членів рівняння.
2. Чим відрізняється задача класифікації від регресії?
3. Яким способом можна покращити якість окремого дерева рішень?
4. Які класи PySpark необхідно використати для навчання моделей регресії та класифікації?
5. За допомогою використання якої функції можна здійснити розділення даних на тренувальну та тестову вибірки? Який сенс такого розділення?
6. Наведіть кілька прикладів задач класифікації та регресії, що ви зустрічали (окрім наведених в даному розділі).
7. Які способи оцінки якості моделей регресії та класифікації ви знаєте?
8. Навчіть модель регресії використовуючи інший пакет машинного навчання. Порівняйте результати. Чи отримали ви подібну якість?

Практична робота №3 – Використання нереляційної БД для зберігання та обробки великих масивів даних

Мета роботи: закріплення навичок роботи із нереляційною БД MongoDB. Створення таблиць, зберігання, обробка та агрегування даних в нереляційній БД в Python.

Практична робота присвячена ознайомленню із роботою з нереляційною базою даних MongoDB. В ході виконання практичної роботи здобувач закріпить навички створення запитів фільтрування, сортування даних, об'єднання декількох таблиць, створення та видалення елементів із використанням мови програмування Python та драйвера БД pymongo. Для розв'язання задачі практичної роботи за узгодженням з викладачем студент може запропонувати свій набір даних.

3.1. Теоретичні відомості

MongoDB – потужний інструмент, з яким легко почати роботу. У цьому розділі ми познайомимося з деякими основними поняттями MongoDB:

- *документ* є основною одиницею даних у MongoDB і приблизно еквівалентний рядку в реляційній системі керування базами даних (але набагато виразніший);
- аналогічно *колекцію* можна розглядати як таблицю з динамічною схемою;
- один примірник MongoDB може містити кілька незалежних *баз даних*, кожна з яких містить свої власні колекції;
- кожен документ має спеціальний ключ «_id», що є унікальним у межах колекції;
- MongoDB поширюється за допомогою простого, але потужного інструменту під назвою оболонка *mongo*. Оболонка *mongo shell* надає вбудовану підтримку для адміністрування екземплярів MongoDB і маніпулювання даними з використанням мови запитів MongoDB. Це також повнофункціональний інтерпретатор JavaScript, який дає змогу користувачам створювати та завантажувати власні сценарії для різних цілей.

Документи. В основі MongoDB лежить документ: упорядкований набір ключів зі зв'язаними значеннями. Подання документа залежить від мови програмування, але більшість мов мають природну структуру даних, наприклад асоціативний масив або словник. Наприклад, у JavaScript документи представлені у вигляді об'єктів:

```
{"greeting" : "Hello, world!"}
```

Цей простий документ містить єдиний ключ `"greeting"` зі значенням `"Hello, world!"`. Більшість документів будуть складнішими порівняно з цим і часто міститимуть кілька пар типу «ключ/значення»:

```
{"greeting" : "Hello, world!", "views" : 3}
```

Як видно, значення в документах – це не просто «BLOB-об'єкти». Вони можуть належати до одного з декількох типів даних (або навіть до всього вкладеного документа). У цьому прикладі значення `"greeting"` є рядком, тоді як значення `"views"` – це ціле число.

Ключі в документі – це рядки. У ключі допустиме використання будь-якого символу в кодуванні UTF-8, за кількома помітними винятками: символи . і $ володіють деякими спеціальними властивостями і повинні використовуватися тільки за певних обставин. Загалом їх слід вважати зарезервованими, і драйвери скаржитимуться, якщо ці символи будуть використовуватися не за призначенням.

MongoDB чутлива до типу і регістру. Наприклад, ці документи відрізняються:

```
{"count" : 5}
{"count" : "5"}
```

так само як і ці:

```
{"count" : 5}
{"Count" : 5}
```

І ще одна важлива річ: документи в MongoDB не можуть містити дублікати ключів. Наприклад, наведений нижче документ не є допустимим:

```
{"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"}
```

Колекції. *Колекція* являє собою групу документів. Якщо в MongoDB документ є аналогом рядка в реляційній СУБД, то колекцію можна розглядати як аналог таблиці.

Динамічні схеми. Колекції мають динамічні схеми. Це означає, що документи в одній колекції можуть мати будь-яку кількість різних «фігур». Наприклад, обидва наведені нижче документи можуть зберігатися в одній колекції:

```
{"greeting" : "Hello, world!", "views": 3}
{"signoff": "Good night, and good luck"}
```

Зверніть увагу на те, що попередні документи мають різні ключі, різну кількість ключів і значення різних типів. Оскільки будь-який документ можна помістити в будь-яку колекцію, часто виникає запитання: «Навіщо взагалі потрібні окремі колекції?» Якщо немає потреби в окремих схемах для різних видів документів, навіщо використовувати додаткові колекції? На те є низка вагомих причин:

– зберігання різних видів документів в одній колекції може стати кошмаром для розробників і адміністраторів. Розробники повинні переконатися, що кожен запит повертає тільки документи, прив'язані до певної схеми, або що код програми, який виконує запит, може обробляти

документи різної форми. Якщо ми запитуємо пости в блозі, дуже складно відсіяти документи, що містять дані про авторів;
- отримати список колекцій набагато швидше, ніж витягти список типів документів у колекції. Наприклад, якби в кожному документі було поле «type», в якому вказувалося, чи був це документ «персонал», «платежі» або «інший», було б набагато повільніше шукати ці три значення в одній колекції, ніж мати три окремі колекції та запитати правильну;
- угруповання документів одного і того ж виду в одній колекції допускає локальність даних. Отримання кількох постів у блозі з колекції, що містить тільки пости, імовірно, потребуватиме менше операцій пошуку на диску, ніж отримання тих самих постів із колекції, де містяться пости і дані про авторів;
- ми починаємо нав'язувати своїм документам певну структуру при створенні індексів. (Це особливо вірно у випадку з унікальними індексами.) Ці індекси визначаються для кожної колекції. Поміщаючи в одну колекцію тільки документи одного типу, можна ефективніше індексувати свої колекції.

Існують вагомі причини для створення схеми та групування пов'язаних типів документів. Хоча цього і не потрібно за замовчуванням, визначення схем для вашого додатка є гарною практикою і може бути реалізовано за допомогою функцій перевірки документації MongoDB і бібліотек об'єктно-документного відображення, доступних для безлічі мов програмування.

Іменування. Колекція ідентифікується за ім'ям. Імена колекцій можуть бути будь-яким рядком у кодуванні UTF-8 з деякими обмеженнями:
- порожній рядок («») не є допустимим ім'ям колекції;
- не слід створювати колекції з іменами, що починаються зі слова system. Цей префікс зарезервовано для внутрішніх колекцій. Наприклад, колекція system.users містить користувачів бази даних, а колекція system.namespaces містить інформацію про всі колекції бази даних;
- створені користувачем колекції не повинні містити зарезервований символ $ у своїх іменах. Різні драйвери, доступні для бази даних, все ж підтримують застосування цього символу в іменах колекцій, тому що він міститься в деяких згенерованих системою колекціях, але ви не повинні використовувати $ в імені, тільки якщо ви не виконуєте доступ до однієї з цих колекцій.

Вкладені колекції. Одна з угод для організації колекцій полягає в тому, щоб використовувати вкладені колекції простору імен, розділені символом «.». Наприклад, додаток, що містить блог, може мати колекцію з ім'ям blog.posts і окрему колекцію з ім'ям blog.authors. Це слугує лише організаційним цілям – немає жодного зв'язку між колекцією blog (вона навіть не повинна існувати) та її «нащадками». Хоча вкладені колекції не мають якихось спеціальних

властивостей, вони корисні і включені в багато інструментів MongoDB. Наприклад:

- GridFS, протокол для зберігання великих файлів, використовує вкладені колекції для зберігання метаданих файлів окремо від блоків вмісту;
- більшість драйверів надають певний «синтаксичний цукор» для доступу до вкладеної колекції. Наприклад, в оболонці бази даних db.blog надасть вам колекцію blog, а db.blog.posts – колекцію blog.posts.

Вкладені колекції – хороший спосіб організувати дані в MongoDB для безлічі випадків використання.

Бази даних. На додаток до групування документів за колекцією MongoDB групує колекції в бази даних. Один екземпляр MongoDB може містити кілька баз даних, кожна з яких об'єднує нуль або більше колекцій. Є гарне практичне правило – зберігати всі дані одного застосунку в одній і тій самій базі даних. Окремі бази даних корисні при зберіганні даних для декількох додатків або користувачів на одному сервері MongoDB.

Як і колекції, бази даних ідентифікуються за ім'ям. Імена баз даних можуть бути будь-яким рядком у форматі UTF-8 з такими обмеженнями:

- порожній рядок («») не є допустимим ім'ям бази даних; - ім'я бази даних не може містити такі символи: /, \, ., », *, , :, |, ?, $, (один пробіл). Здебільшого дотримуватися буквено-цифрової таблиці ASCII;
- імена баз даних нечутливі до регістру;
- імена баз даних обмежені максимум 64 байтами.

Традиційно, до використання підсистеми зберігання WiredTiger, імена баз даних ставали файлами у вашій файловій системі. Тепер цього більше немає. Це пояснює, чому багато які з попередніх обмежень взагалі існують.

Є також деякі зарезервовані імена баз даних, до яких можна отримати доступ, але які мають особливу семантику. Ось вони:

- *admin* – база даних *admin* відіграє роль в автентифікації та авторизації. Крім того, доступ до цієї бази даних необхідний для низки адміністративних операцій.
- *local* – у цій базі даних зберігаються дані, що належать до одного сервера. У наборах реплік у базі *local* зберігаються дані, що використовуються в процесі реплікації. Сама база даних *local* ніколи не реплікується.
- *config* – розділені (сегментовані) кластери MongoDB використовують базу даних *config* для зберігання інформації про кожен шард.

Об'єднуючи ім'я бази даних із колекцією в цій базі даних, ви можете отримати повне ім'я колекції, яке називається простором імен. Наприклад, якщо ви використовуєте колекцію blog.posts у базі даних cms, простір імен цієї колекції буде таким: cms.blog.posts. Довжина просторів імен обмежена 120 байтами, а на практиці має бути менше 100 байт.

3.2. Приклад розв’язування задачі

Першим кроком необхідно завантажити MongoDB за посиланням (рис. 3.1). Далі йде стандартна процедура інсталяції із стандартними налаштуваннями (рис. 3.2).

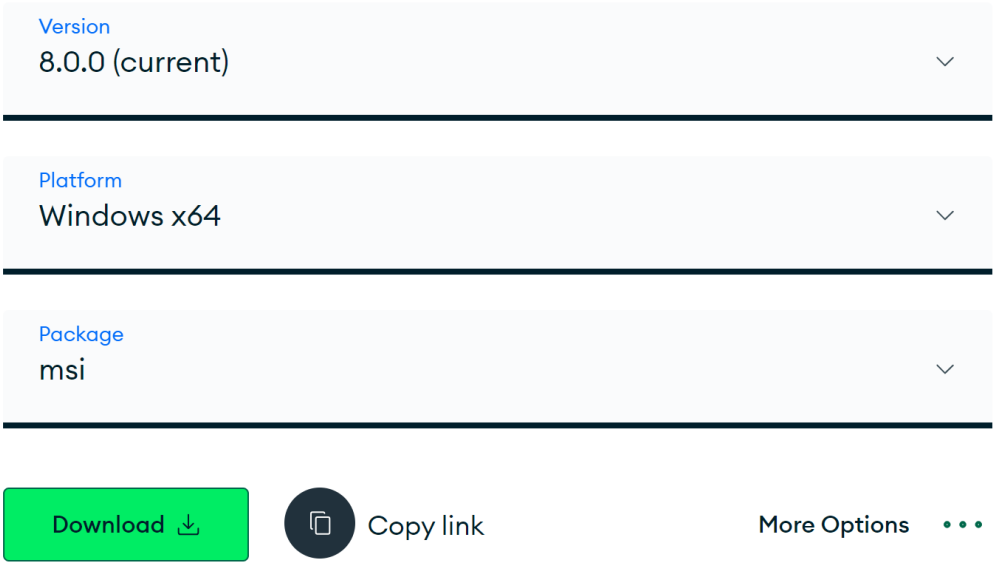

Рис. 3.1. Завантаження MongoDB.

Рис. 3.2. Початок встановлення MongoDB.

Для роботи з MongoDB із мовою програмування Python необхідно встановити відповідний пакет:

Командний рядок

```
pip install pymongo
```

Створення баз даних та колекцій

Створення бази даних. Щоб створити базу даних у MongoDB, почніть із створення об'єкта MongoClient, а потім укажіть URL-адресу з'єднання з правильною ip-адресою та назвою бази даних, яку ви хочете створити.

MongoDB створить базу даних, якщо вона не існує, і підключиться до неї.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["mydatabase"]
```

Важливо: у MongoDB база даних не створюється, доки вона не отримає вміст!

MongoDB чекає, доки ви не створите колекцію (таблицю) з принаймні одним документом (записом), перш ніж фактично створити базу даних (і колекцію).

Перевірка існування бази даних

Пам'ятайте: у MongoDB база даних не створюється, доки вона не отримає вміст, тому, якщо ви створюєте базу даних уперше, вам слід завершити наступні два розділи (створення колекції та створення документа), перш ніж перевіряти, чи існує база даних!

Ви можете перевірити, чи існує база даних, перерахувавши всі бази даних у вашій системі:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

print(myclient.list_database_names())
```

Вивід

```
['admin', 'config', 'local']
```

Створення колекції. Колекція в MongoDB — це те саме, що таблиця в базах даних SQL. Щоб створити колекцію в MongoDB, використовуйте об'єкт бази даних і вкажіть назву колекції, яку ви хочете створити.

MongoDB створить колекцію, якщо вона не існує.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

mycol = mydb["customers"]
```

Важливо: у MongoDB колекція не створюється, доки вона не отримає вміст! MongoDB чекає, поки ви вставите документ, перш ніж фактично створити колекцію.

Перевірте, чи існує колекція. Пам'ятайте: у MongoDB колекція не створюється, доки вона не отримає вміст, тому, якщо ви створюєте колекцію вперше, вам слід завершити наступний розділ (створити документ), перш ніж перевіряти, чи існує колекція!

Ви можете перевірити, чи існує колекція в базі даних, перерахувавши всі колекції:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

print(mydb.list_collection_names())
```

Додавання даних

Вставлення в колекцію

Документ у MongoDB – це те саме, що запис у базах даних SQL.

Щоб вставити запис або документ, як це називається в MongoDB, у колекцію, ми використовуємо метод insert_one().

Перший параметр методу insert_one() – це словник, що містить імена та значення кожного поля в документі, який потрібно вставити.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydict = { "name": "John", "address": "Highway 37" }

x = mycol.insert_one(mydict)
```

Поле _id. Метод insert_one() повертає об'єкт InsertOneResult, який має властивість inserted_id, яка містить ідентифікатор вставленого документа.

Вставте інший запис у колекцію «клієнти» та поверніть значення поля _id:

Код

```
mydict = { "name": "Peter", "address": "Lowstreet 27" }

x = mycol.insert_one(mydict)

print(x.inserted_id)
```

Вивід

```
66fadf5465f84ef9c38a7b94
```

Якщо ви не вкажете поле _id, MongoDB додасть його для вас і призначить унікальний ідентифікатор для кожного документа.

У наведеному вище прикладі не було вказано поле _id, тому MongoDB призначив унікальний _id для запису (документа).

Вставлення декількох документів. Щоб вставити кілька документів у колекцію в MongoDB, ми використовуємо метод insert_many().

Перший параметр методу insert_many() – це список словників із даними, які потрібно вставити:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)
```

```
# Друк ідентифікаторів _id вставлених документів:
print(x.inserted_ids)
```

Вивід

```
[ObjectId('66fadfa8266aaa9f6110e31f'), ObjectId('66fadfa8266aaa9f6110e320'),
ObjectId('66fadfa8266aaa9f6110e321'), ObjectId('66fadfa8266aaa9f6110e322'),
ObjectId('66fadfa8266aaa9f6110e323'), ObjectId('66fadfa8266aaa9f6110e324'),
ObjectId('66fadfa8266aaa9f6110e325'), ObjectId('66fadfa8266aaa9f6110e326'),
ObjectId('66fadfa8266aaa9f6110e327'), ObjectId('66fadfa8266aaa9f6110e328'),
ObjectId('66fadfa8266aaa9f6110e329'), ObjectId('66fadfa8266aaa9f6110e32a')]
```

Метод insert_many() повертає об’єкт InsertManyResult, який має властивість inserted_ids, яка містить ідентифікатори вставлених документів.

Вставлення кількох документів із зазначеними ідентифікаторами. Якщо ви не хочете, щоб MongoDB призначав унікальні ідентифікатори для вашого документа, ви можете вказати поле _id під час вставлення документа(ів).

Пам’ятайте, що значення мають бути унікальними. Два документи не можуть мати однаковий _id.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)
```

```
# Друк ідентифікаторів _id вставлених документів:
print(x.inserted_ids)
```

Вивід
`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]`

Імпорт даних із стороннього джерела. Також зручним інструментом для імпорту даних є середовище MongoDB Compass, що автоматично встановилось із MongoDB.

Після відкриття MongoDB Compass необхідно створити з'єднання із БД. Для цього натискаємо «Add New Connection» (рис. 3.3).

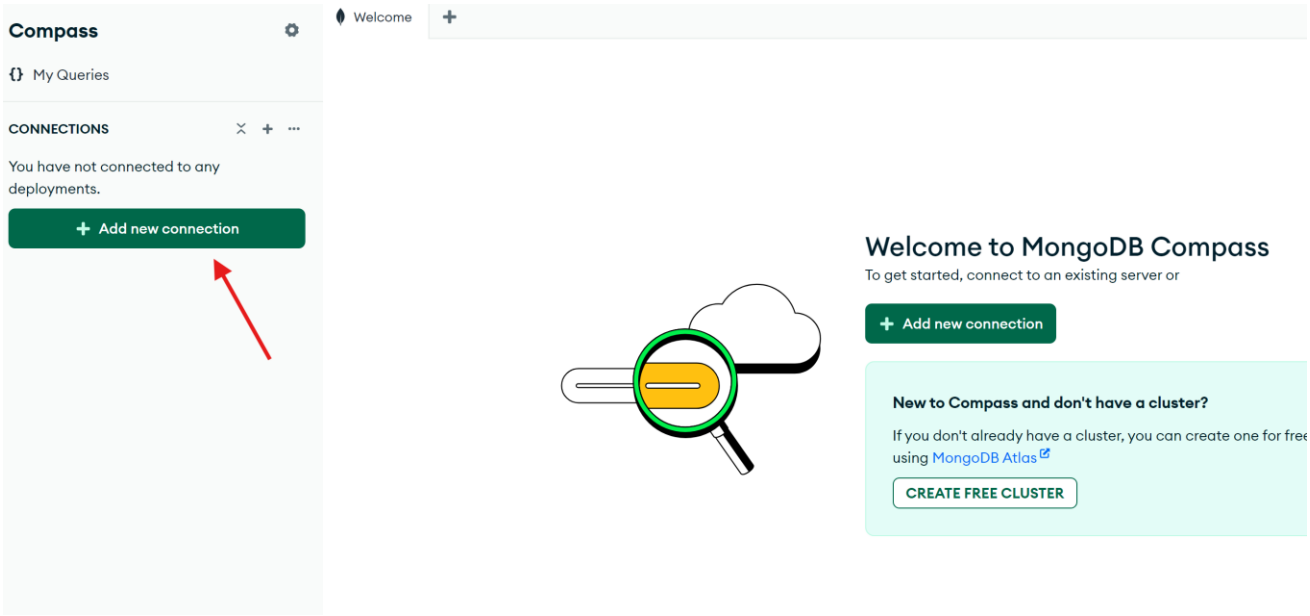

Рис. 3.3. Створення нового з'єднання з БД.

У вікні «Add New Connection» залишаємо всі налаштування без змін та натискаємо «Connect» (рис. 3.4).

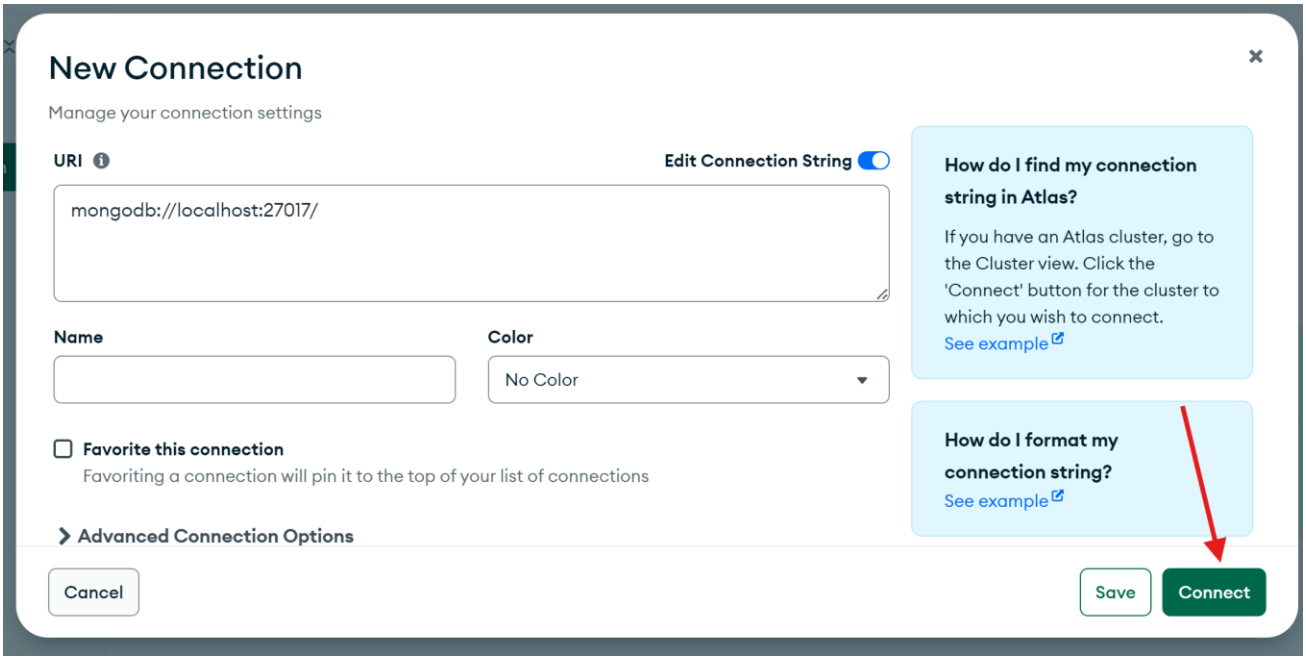

Рис. 3.4. З'єднання з БД.

Натиснувши «+» можна створити нову базу даних «MyDb» та колекцію в ній «MyCollection» (рис. 3.5 та рис. 3.6).

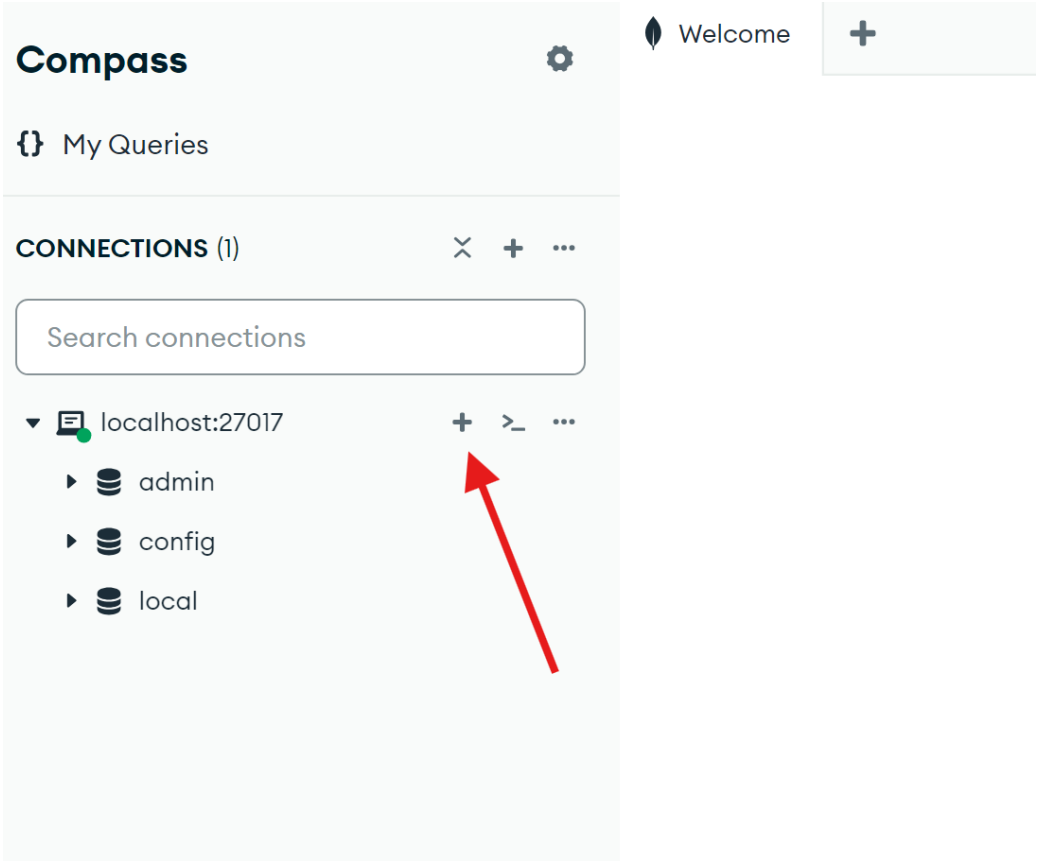

Рис. 3.5. Кнопка для створення нової бази даних.

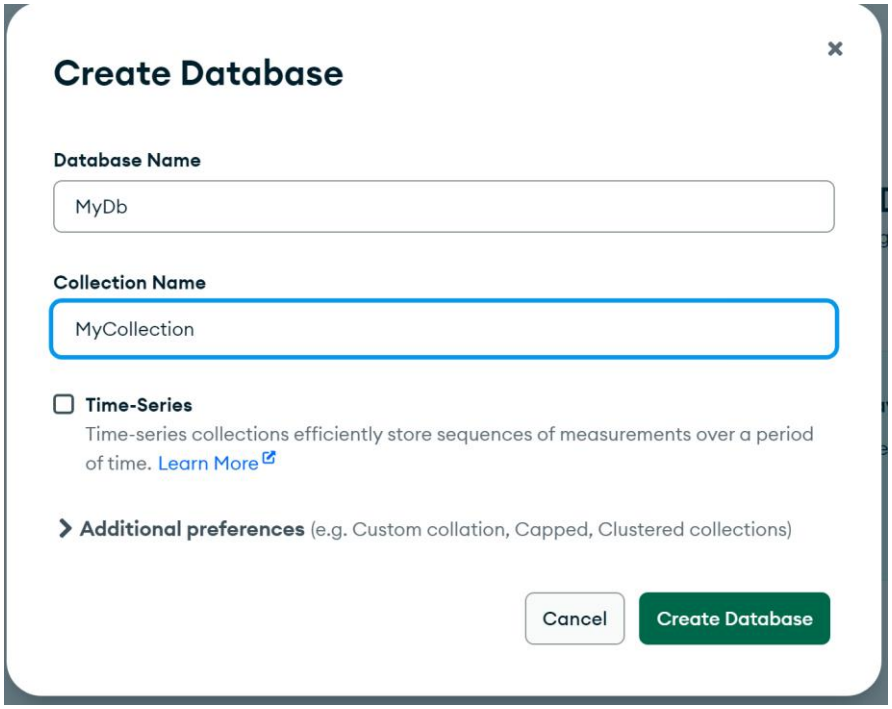

Рис. 3.6. Створення бази даних та колекції в ній.

Далі для імпорту даних натискаємо «Add Data» і після цього «Import JSON or CSV file» (рис. 3.7). Обираємо файл із готовими даними. У вікні, що з'явилось, натискаємо «Import» для завершення імпорту. Після цього, ви отримаєте колекцію бази даних із доданими документами з обраного файлу.

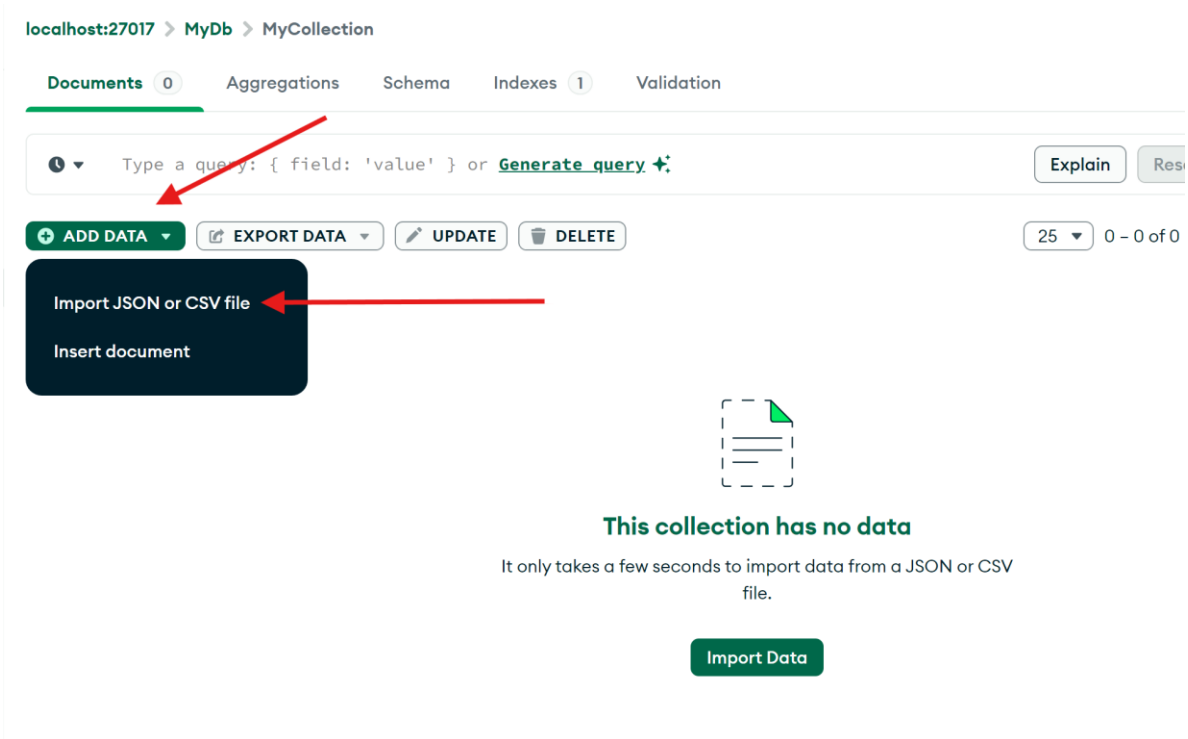

Рис. 3.7. Додавання даних.

Операції пошуку даних

У MongoDB ми використовуємо методи find() і find_one() для пошуку даних у колекції.

Так само, як оператор SELECT використовується для пошуку даних у таблиці в базі даних MySQL.

Пошук одного елемента. Щоб вибрати дані з колекції в MongoDB, ми можемо використати метод find_one().

Метод find_one() повертає перше входження у вибірку.

Знайдіть перший документ у колекції клієнтів:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.find_one()

print(x)
```

Вивід

```
{'_id': ObjectId('66fadf27e360b98a5cd8b009'), 'name': 'John', 'address': 'Highway
37'}
```

Знайти всі елементи. Щоб вибрати дані з таблиці в MongoDB, ми також можемо використовувати метод find().

Метод find() повертає всі елементи.

Першим параметром методу find() є об’єкт запиту. У цьому прикладі ми використовуємо порожній об’єкт запиту, який вибирає всі документи в колекції.

Метод find() без параметрів є еквівалентним SELECT * в SQL.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find():
  print(x)
```

Вивід

```
{‘_id’: 1, ‘name’: ‘John’, ‘address’: ‘Highway 37’}
{‘_id’: 2, ‘name’: ‘Peter’, ‘address’: ‘Lowstreet 27’}
{‘_id’: 3, ‘name’: ‘Amy’, ‘address’: ‘Apple st 652’}
{‘_id’: 4, ‘name’: ‘Hannah’, ‘address’: ‘Mountain 21’}
{‘_id’: 5, ‘name’: ‘Michael’, ‘address’: ‘Valley 345’}
{‘_id’: 6, ‘name’: ‘Sandy’, ‘address’: ‘Ocean blvd 2’}
(вивід скорочено)
```

Запити до даних

Фільтрування результату. Під час пошуку документів у колекції ви можете відфільтрувати результат за допомогою об’єкта запиту.

Перший аргумент методу find() є об’єктом запиту, який використовується для обмеження пошуку.

Знайти документ(и) з адресою «Park Lane 38»:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Park Lane 38" }

mydoc = mycol.find(myquery)

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': ObjectId('66fadfa8266aaa9f6110e327'), 'name': 'Ben', 'address': 'Park Lane
38'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
```

Розширений запит. Щоб робити розширені запити, ви можете використовувати модифікатори як значення в об’єкті запиту.

наприклад щоб знайти документи, у яких поле «адреса» починається з літери «S» або вище (за алфавітом), використовуйте модифікатор «більше ніж»: {"$gt": "S"}:

Знайдіть документи, адреса яких починається з літери "S" або вище:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$gt": "S" } }

mydoc = mycol.find(myquery)

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': ObjectId('66fadfa8266aaa9f6110e321'), 'name': 'Michael', 'address':
'Valley 345'}
{'_id': ObjectId('66fadfa8266aaa9f6110e324'), 'name': 'Richard', 'address': 'Sky
st 331'}
{'_id': ObjectId('66fadfa8266aaa9f6110e326'), 'name': 'Vicky', 'address': 'Yellow
Garden 2'}
{'_id': ObjectId('66fadfa8266aaa9f6110e32a'), 'name': 'Viola', 'address': 'Sideway
1633'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

Сортування результатів

Використовуйте метод sort() для сортування результату за зростанням або спаданням. Метод sort() отримує один параметр «назва поля» і один параметр «напрямок» (за замовчуванням напрямок за зростанням).

Сортування результату в алфавітному порядку за назвою:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name")

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': ObjectId('66fadfa8266aaa9f6110e31f'), 'name': 'Amy', 'address': 'Apple st
652'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('66fadfa8266aaa9f6110e327'), 'name': 'Ben', 'address': 'Park Lane
38'}
{'_id': ObjectId('66fadfa8266aaa9f6110e323'), 'name': 'Betty', 'address': 'Green
Grass 1'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('66fadfa8266aaa9f6110e329'), 'name': 'Chuck', 'address': 'Main
Road 989'}
{'_id': ObjectId('66fadfa8266aaa9f6110e320'), 'name': 'Hannah', 'address':
'Mountain 21'}
(вивід скорочено)
```

Сортування за спаданням. Використовуйте значення -1 як другий параметр для сортування за спаданням.

```
sort("name", 1) # за зростанням
sort("name", -1) # за спаданням
```

Сортування результату у зворотному порядку за алфавітом за назвою:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name", -1)

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('66fadfa8266aaa9f6110e328'), 'name': 'William', 'address':
'Central st 954'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': ObjectId('66fadfa8266aaa9f6110e32a'), 'name': 'Viola', 'address': 'Sideway
1633'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('66fadfa8266aaa9f6110e326'), 'name': 'Vicky', 'address': 'Yellow
Garden 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
```

```
{'_id': ObjectId('66fadfa8266aaa9f6110e325'), 'name': 'Susan', 'address': 'One way
98'}
{'_id': ObjectId('66fadfa8266aaa9f6110e322'), 'name': 'Sandy', 'address': 'Ocean
blvd 2'}
(вивід скорочено)
```

Видалення даних

Видалення документа. Щоб видалити один документ, ми використовуємо метод delete_one().

Перший параметр методу delete_one() — це об'єкт запиту, який визначає, який документ потрібно видалити.

Примітка: якщо запит знаходить більше одного документа, видаляється лише перший екземпляр.

Видалити документ з адресою «Mountain 21»:

Код

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Mountain 21" }

mycol.delete_one(myquery)
```

Вивід

Видалення багатьох документів. Щоб видалити кілька документів, скористайтеся методом delete_many().

Перший параметр методу delete_many() — це об'єкт запиту, який визначає, які документи потрібно видалити.

Видалити всі документи, адреса яких «Mountain 21»:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Yellow Garden 2" }

x = mycol.delete_many(myquery)
```

```
print(x.deleted_count, " documents deleted.")
```

Вивід

```
2  documents deleted.
```

Видалити всі документи в колекції. Щоб видалити всі документи в колекції, передайте порожній об’єкт запиту методу delete_many():

Видалити всі документи в колекції «customers»:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.delete_many({})

print(x.deleted_count, " documents deleted.")
```

Вивід

```
25 documents deleted.
```

3.3. Індивідуальне завдання

1. Завантажити дані за індивідуальним варіантом за посиланням на порталі ДО. Здійснити їх імпорт з використанням MongoDB Compass. Вивести дані в Python та показати перші 5 рядків.

Далі виконати наступні завдання:

№	Текст завдання
1	2. Знайти всі товари з продавцем «Seller1». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за описом. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
2	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
3	2. Знайти всі товари з категорії «Clothing».

	3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за категорією. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
4	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
5	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари категорії «Materials» з ціною менше 400. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
6	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари категорії «Food» з ціною більше 120. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
7	2. Знайти всі товари з продавцем «Seller1». 3. Знайти всі товари категорії «Food» з ціною більше 120. 4. Відсортувати дані за спаданням назви продавця. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
8	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за спаданням назви продавця. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
9	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за описом. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
10	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари категорії «Food» з ціною більше 120. 4. Відсортувати дані за категорією. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
11	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
12	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари категорії «Materials» з ціною менше 400. 4. Відсортувати дані за спаданням назви продавця. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
13	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари з назвою «Wood» та ціною менше 20.

	4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
14	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари категорії «Materials» з ціною менше 400. 4. Відсортувати дані за описом. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
15	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
16	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за описом. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
17	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
18	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
19	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
20	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари категорії «Materials» з ціною менше 400. 4. Відсортувати дані за категорією. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
21	2. Знайти всі товари з продавцем «Seller1». 3. Знайти всі товари категорії «Materials» з ціною менше 400. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
22	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари категорії «Food» з ціною більше 120. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
23	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за описом.

	5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
24	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
25	2. Знайти всі товари з назвою «Apple iPhone». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за категорією. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
26	2. Знайти всі товари з категорії «Electronics». 3. Знайти всі товари категорії «Materials» з ціною менше 400. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
27	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари з назвою «Wood» та ціною менше 20. 4. Відсортувати дані за спаданням назви. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
28	2. Знайти всі товари з продавцем «Seller1». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за спаданням ціни. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
29	2. Знайти всі товари з продавцем «Seller1». 3. Знайти всі товари категорії «Food» з ціною більше 120. 4. Відсортувати дані за категорією. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).
30	2. Знайти всі товари з категорії «Clothing». 3. Знайти всі товари з назвою «Eggs» та ціною менше 15. 4. Відсортувати дані за описом. 5. Додати довільний документ подібний за структурою до вже існуючих. Показати код додавання. Написати запит для знаходження цього документа. Видалити даний документ (і лише його).

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, по кожному запиту: скріншот середовища, код та вивід запиту.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

3.4. Контрольні питання

1. Зазначте основні структурні елементи нереляційної БД MongoDB. Зазначте відповідні терміни для реляційних БД.
2. Який пакет Python необхідно використати для під’єднання до MongoDB?
3. Що таке «драйвер» в термінах MongoDB?
4. Які особливості створення БД та колекцій в MongoDB?
5. Як здійснити пошук елемента в MongoDB? Як вказати, що потрібен більший або менший елемент?
6. У який спосіб зручно імпортувати дані із стороннього джерела?
7. Як відсортувати дані за спаданням певного поля?
8. Що повертає функція видалення елементів в MongoDB?

Практична робота №4 – Інтеграція PySpark і MongoDB

Мета роботи: узагальнення навичок, отриманих в курсі. Поєднання різних технологій для обробки, структурування, аналізу та передбачення на даних.

Практична робота присвячена знайомству з ETL підходом, інтеграції PySpark та MongoDB. В ході виконання практичної роботи здобувач навчиться будувати конвеєри вилучення та обробки даних з різних джерел та подальшого їх збереження в центральній базі даних. Для розв'язання задачі практичної роботи за узгодженням з викладачем студент може запропонувати свій набір даних.

4.1. Теоретичні відомості

Що таке ETL? ETL (Extract, Transform, Load) – означає «витягнути, перетворити, завантажити» – це процес інтеграції даних, який об'єднує, очищає та організовує дані з декількох джерел в єдиний, узгоджений набір даних для зберігання в сховищі даних або іншій цільовій системі.

Конвеєри даних ETL забезпечують основу для аналізу даних і робочих потоків машинного навчання. За допомогою низки бізнес-правил ETL очищає та організовує дані для задоволення конкретних потреб бізнес-аналітики, таких як щомісячна звітність, але вона також може вирішувати більш складні аналітичні завдання, які можуть покращити внутрішні процеси та досвід кінцевих користувачів. Конвеєри ETL часто використовуються організаціями для:

- вилучення даних із застарілих систем;
- очищення даних для покращення якості та узгодженості даних;
- завантаження даних у цільову базу даних.

Як розвивався ETL. Компанії генерують дані ще з часів рахівниць, але сучасна аналітика стала можливою лише з появою цифрових комп'ютерів і сховищ даних.

Значний крок вперед був зроблений у 1970-х роках, коли відбувся перехід до великих централізованих баз даних. Тоді було запроваджено ETL як процес інтеграції та завантаження даних для обчислень і аналізу, який з часом став основним методом обробки даних для проектів зі створення сховищ даних.

Наприкінці 1980-х років сховища даних і перехід від транзакційних баз даних до реляційних баз даних, які зберігали інформацію у форматах реляційних даних, набули популярності. Старіші транзакційні бази даних зберігали інформацію від транзакції до транзакції, причому дублікати інформації про клієнтів зберігалися з кожною транзакцією, тому не було простого способу отримати доступ до даних про клієнтів в уніфікованому вигляді з плином часу. З появою реляційних баз даних аналітика стала основою бізнес-аналітики (BI) і важливим інструментом для прийняття рішень.

До появи більш досконалого програмного забезпечення ETL, перші спроби були здебільшого ручними зусиллями IT-команди для вилучення даних з різних

систем і з'єднувачів, перетворення даних у загальний формат, а потім завантаження їх у взаємопов'язані таблиці. Тим не менш, перші кроки ETL були варті зусиль, оскільки вдосконалені алгоритми, а також розвиток нейронних мереж створювали дедалі глибші можливості для аналітичних досліджень.

Ера великих даних настала в 1990-х роках, коли швидкість обчислень і ємність сховищ продовжували стрімко зростати, а великі обсяги даних надходили з нових джерел, таких як соціальні мережі та Інтернет речей (IoT). Обмежуючим фактором залишається те, що дані часто зберігаються в локальних сховищах даних.

Наступним важливим кроком як в обчисленнях, так і в ETL були хмарні обчислення, які стали популярними наприкінці 1990-х років. Завдяки таким сховищам даних, як Amazon Web Services (AWS), Microsoft Azure і Snowflake, доступ до даних тепер можна отримати з усього світу і швидко масштабувати, що дозволяє рішенням ETL надавати чудову детальну інформацію та нові конкурентні переваги.

Останнім досягненням є рішення ETL, що використовують потокові дані для отримання оперативної інформації з величезних масивів даних.

ETL в порівнянні з ELT. Найбільш очевидна різниця між ETL і ELT – витягти, завантажити, перетворити – полягає в порядку виконання операцій. ELT копіює або експортує дані з джерела, але замість того, щоб завантажувати їх у проміжну зону для перетворення, він завантажує необроблені дані безпосередньо в цільове сховище даних, де вони будуть перетворені за потреби.

Хоча обидва процеси використовують різні сховища даних, кожен з них має свої переваги і недоліки. ELT корисний для поглинання великих обсягів неструктурованих наборів даних, оскільки завантаження може відбуватися безпосередньо з джерела. ELT може бути більш ідеальним для управління великими даними, оскільки не потребує попереднього планування вилучення та зберігання даних.

Процес ETL вимагає більшої визначеності на самому початку. Потрібно визначити конкретні точки даних для вилучення, а також будь-які потенційні «ключі» для інтеграції між різними системами джерел. Джерело вхідних даних часто відстежується за допомогою метаданих. Навіть після завершення цієї роботи необхідно створити бізнес-правила для перетворення даних. Ця робота зазвичай може залежати від вимог до даних для певного типу аналізу даних, що визначатиме рівень узагальнення, який повинні мати дані.

Хоча конвеєри ELT стають все більш популярними з впровадженням хмарних баз даних, технологія ELT все ще розвивається, а це означає, що найкращі практики все ще формуються.

Як працює ETL. Найпростіший спосіб зрозуміти, як працює ETL, – це зрозуміти, що відбувається на кожному кроці процесу.

Витягування даних (Extract). Під час видобування даних необроблені дані копіюються або експортуються з місця їхнього походження до місця обробки.

Команди управління даними можуть витягувати дані з різних джерел, які можуть бути структурованими або неструктурованими. Ці типи даних включають, але не обмежуються ними:

- SQL або NoSQL сервери;
- CRM і ERP системи;
- JSON та XML;
- плоскі бази даних;
- електронна пошта;
- веб-сторінки.

Перетворення. На етапі обробки вихідні дані проходять обробку. Тут дані трансформуються і консолідуються для цільового аналітичного використання. Ця фаза процесу трансформації може включати:

- фільтрацію, очищення, агрегування, дедуплікацію, перевірку достовірності та автентичності даних;
- виконання розрахунків, перекладів або узагальнень на основі вихідних даних. Це може включати зміну заголовків рядків і стовпців для узгодженості, конвертацію валют або інших одиниць виміру, редагування текстових рядків тощо;
- проведення аудиту для забезпечення якості та відповідності даних, а також обчислення метрик;
- видалення, шифрування або захист даних, що регулюються галузевими або державними регуляторами;
- форматування даних у таблиці або об'єднані таблиці відповідно до схеми цільового сховища даних.

Завантаження. На цьому останньому етапі перетворені дані переміщуються з місця зберігання до цільового сховища даних. Зазвичай це передбачає початкове завантаження всіх даних, а потім періодичне завантаження інкрементних змін даних і, рідше, повне оновлення для стирання і заміни даних у сховищі. Для більшості організацій, які використовують ETL, цей процес є автоматизованим, чітко визначеним, безперервним і пакетним. Зазвичай процес завантаження ETL відбувається в неробочий час, коли трафік у вихідних системах і сховищі даних найнижчий.

4.2. Приклад розв’язування задачі

Приклад: Створення єдиної бази даних клієнтів

Уявімо, що ми – компанія, яка займається електронною комерцією і має кілька інтернет-магазинів у різних регіонах, кожен з яких має власну систему баз даних. Наша мета – створити єдину базу даних клієнтів, яку можна використовувати для аналізу та маркетингу.

Поточний стан:

1. Магазин A: цей магазин використовує базу даних MongoDB для зберігання інформації про клієнтів, включаючи ім'я, електронну пошту, адресу та історію покупок.
2. Магазин B: цей магазин використовує базу даних MySQL для зберігання аналогічної інформації про клієнтів.
3. Магазин C: цей магазин не має централізованої системи баз даних; дані про клієнтів зберігаються в декількох електронних таблицях і CRM-системах.

Проблеми такої системи:

1. Неузгодженість даних: база даних кожного магазину має власну унікальну структуру, що ускладнює інтеграцію даних з різних джерел.
2. Відсутність уніфікованого аналізу: через розрізненість баз даних наша маркетингова команда не може легко отримати доступ та проаналізувати поведінку клієнтів у всіх магазинах.
3. Труднощі з ідентифікацією цінних клієнтів: без централізованого перегляду інформації про клієнтів нам важко ідентифікувати клієнтів з найбільшими витратами або прогнозувати майбутні моделі покупок.

ETL-рішення. Для створення єдиної бази даних клієнтів ми впровадили ETL-конвеєр, який витягує дані з бази даних кожного магазину, перетворює їх у стандартизований формат і завантажує в центральну колекцію MongoDB. Це дозволяє нам:

1. Уніфікувати інформацію про клієнтів: ми витягуємо відповідні поля (наприклад, ім'я, електронну пошту, адресу) з усіх баз даних і перетворюємо їх у єдиний формат.
2. Агрегувати історію покупок: ми агрегуємо дані про покупки в усіх магазинах, щоб створити повне уявлення про купівельну поведінку кожного клієнта.
3. Створення єдиної уніфікованої бази даних: конвеєр ETL завантажує перетворені дані в нову колекцію в нашій базі даних MongoDB, що робить її легко доступною для аналізу та маркетингових цілей.

Переваги ETL:

1. Уніфікований аналіз: наша команда маркетологів тепер має доступ до єдиного уніфікованого перегляду інформації про клієнтів у всіх магазинах.
2. Покращене таргетування: визначаючи цінних клієнтів і прогнозуючи майбутні моделі покупок, ми можемо адаптувати наші маркетингові кампанії для кращого залучення цих цінних клієнтів.
3. Удосконалене прийняття рішень на основі даних: маючи всебічне розуміння поведінки клієнтів, ми можемо приймати більш обґрунтовані рішення щодо товарних пропозицій, стратегій ціноутворення та розподілу ресурсів.

Етапи конвеєру ETL:

1. Витяг: ми витягуємо дані про клієнтів з бази даних MongoDB магазину A, бази даних MySQL магазину B та інтегруємо їх з CRM-системою, що використовується магазином C.
2. Перетворення: ми перетворюємо витягнуті дані в стандартизований формат за допомогою PySpark, агрегуючи історію покупок і обчислюючи середню вартість замовлення.
3. Завантаження: перетворені дані завантажуються в нашу центральну колекцію MongoDB, створюючи єдину базу даних клієнтів.

Впровадивши це ETL-рішення, ми створили єдину базу даних клієнтів, яка дозволяє нам приймати більш обґрунтовані рішення щодо маркетингових стратегій і товарних пропозицій, що в кінцевому підсумку сприяє зростанню бізнесу.

Для прикладу розглянемо побудову ETL конвеєру із PySpark для бази даних клієнтів магазину C.

1. Розглянемо дані. Інформацію про клієнтів розділено на файли customers2023.csv та customers2024.csv.

customers2023.csv

```
"id","name","address","subscription"
1,"John Smith","123 Main St, Anytown, CA 12345",19.99
2,"Jane Doe","456 Elm St, Othertown, NY 67890",9.99
3,"Bob Johnson","789 Oak St, Hometown, IL 54321",29.99
4,"Maria Rodriguez","321 Pine St, Smalltown, TX 90123",0.0
5,"David Lee","901 Maple St, Bigcity, FL 34567",0.0
6,"Emily Chen","234 Cedar St, Suburbiatown, OH 45678",24.99
7,"Michael Brown","567 Spruce St, Ruraltown, GA 89012",39.99
```

customers2024.csv

```
"id","name","address","subscription"
8,"Emily Patel","987 Walnut St, Apt 3, Springfield, IL 62704",14.99
10,"Sophia Lee","1234 Oakwood Dr, Sunnyvale, CA 94087",24.99
13,"Jackson Hall","456 Elm St, Othertown, TX 76543",29.99
14,"Isabella Garcia","321 Pine St, Smalltown, FL 32456",39.99
3,"Bob Johnson","789 Oak St, Hometown, IL 54321",29.99
4,"Maria Rodriguez","321 Pine St, Smalltown, TX 90123",13.0
5,"David Lee","901 Maple St, Bigcity, FL 34567",12.0
```

2. Імпортуємо необхідні для роботи пакети

Код

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import lit
```

3. Створимо сесію PySpark. Сконфігуруємо під'єднання до нашої БД MongoDB.

Код

```
spark = SparkSession.builder.appName("Application")
spark = spark.config("spark.mongodb.output.uri",
"mongodb://mongodb:27017/productdb")
spark = spark.config("spark.jars.packages", "org.mongodb.spark:mongo-spark-
connector_2.12:10.4.0")
spark = spark.getOrCreate()
```

4. Завантажимо дані в PySpark.

Код

```
customers2023 = spark.read.csv('customers2023.csv', header=True, inferSchema=True)
customers2024 = spark.read.csv('customers2024.csv', header=True, inferSchema=True)
```

5. Створимо колонку, що визначає рік реєстрації клієнта (в залежності від назви файлу):

Код

```
customers2023 = customers2023.withColumn('year', lit(2023))
customers2024 = customers2024.withColumn('year', lit(2024))
```

6. Об'єднаємо отримані кадри даних.

Код

```
customers = customers2023.union(customers2024)
```

7. Приберемо дублікати. Для клієнтів з однаковим id оберемо максимальну вартість підписки.

Код

```
customers = customers.groupBy(customers.id).max()
```

8. Відобразимо отриманий кадр даних.

Код

```
customers.show()
```

Вивід

```
+---+-------+-----------------+---------+
| id|max(id)|max(subscription)|max(year)|
+---+-------+-----------------+---------+
|  1|      1|            19.99|     2023|
|  6|      6|            24.99|     2023|
|  3|      3|            29.99|     2024|
|  5|      5|             12.0|     2024|
|  4|      4|             13.0|     2024|
|  7|      7|            39.99|     2023|
|  2|      2|             9.99|     2023|
| 13|     13|            29.99|     2024|
|  8|      8|            14.99|     2024|
| 10|     10|            24.99|     2024|
| 14|     14|            39.99|     2024|
+---+-------+-----------------+---------+
```

9. Подальший аналіз показав, що інтереси клієнтів було записано в окремий файл customer_interests.csv (зміст файлу показано нижче). Додамо їх в поточний кадр даних.

customer_interests.csv

```
id,interests
1,"Reading, Music, Traveling"
2,"Cooking, Photography, Writing"
3,"Gaming, Theater, Painting, Yoga"
4,"Dancing, Karaoke, Sports, Art"
5,"Sports, Art, Traveling, Hiking"
6,"Yoga, Meditation, Reading, Writing"
7,"Swimming, Kayaking, Cycling, Biking"
8,"Painting, Sculpture, Gardening, Hiking"
10,"Painting, Sculpture, Reading, Writing"
13,"Rock Climbing, Swimming, Kayaking"
14,"Surfing, Snowboarding, Cycling, Biking"
```

Код

```
customers_interests = spark.read.csv('customers_interests.csv', header=True,
inferSchema=True)
customers = customers.join(customers_interests, 'id')
customers.show()
```

Вивід

```
+---+-------+----------------+---------+--------------------+
| id|max(id)|max(subscription)|max(year)|           interests|
+---+-------+----------------+---------+--------------------+
|  1|      1|           19.99|     2023|Reading, Music, T...|
|  6|      6|           24.99|     2023|Yoga, Meditation,...|
|  3|      3|           29.99|     2024|Gaming, Theater, ...|
|  5|      5|            12.0|     2024|Sports, Art, Trav...|
|  4|      4|            13.0|     2024|Dancing, Karaoke,...|
|  7|      7|           39.99|     2023|Swimming, Kayakin...|
|  2|      2|            9.99|     2023|Cooking, Photogra...|
| 13|     13|           29.99|     2024|Rock Climbing, Sw...|
|  8|      8|           14.99|     2024|Painting, Sculptu...|
| 10|     10|           24.99|     2024|Painting, Sculptu...|
| 14|     14|           39.99|     2024|Surfing, Snowboar...|
+---+-------+----------------+---------+--------------------+
```

10. Також виявилось, що ідентифікатори клієнтів не відповідають формату основної бази даних. Зчитаємо файл customer_ids.csv для перетворення ідентифікаторів

customer_ids.csv

```
id,new_id
1,5cafe5f8e6a0a21d4b2a8f7c
2,5cae9b3ce6a0a21d4b2a8f76
3,5cd0ff8de6a0a21d4b2a8f74
4,5cbaf8e1e6a0a21d4b2a8f7b
5,5cc9c0e4e6a0a21d4b2a8f79
6,5cd3bbaee6a0a21d4b2a8f75
7,5caed1cbe6a0a21d4b2a8f78
8,5ce3e9a4e6a0a21d4b2a8f77
10,5cb8c4b3e6a0a21d4b2a8f7a
13,5cd1db45e6a0a21d4b2a8f72
14,5cf3ba9de6a0a21d4b2a8f7d
```

Код

```
customer_ids = spark.read.csv('customer_ids.csv', header=True, inferSchema=True)
customer_ids.show()
```

Вивід

```
+---+--------------------+
| id|              new_id|
+---+--------------------+
|  1|5cafe5f8e6a0a21d4...|
|  2|5cae9b3ce6a0a21d4...|
|  3|5cd0ff8de6a0a21d4...|
|  4|5cbaf8e1e6a0a21d4...|
|  5|5cc9c0e4e6a0a21d4...|
|  6|5cd3bbaee6a0a21d4...|
|  7|5caed1cbe6a0a21d4...|
|  8|5ce3e9a4e6a0a21d4...|
| 10|5cb8c4b3e6a0a21d4...|
| 13|5cd1db45e6a0a21d4...|
| 14|5cf3ba9de6a0a21d4...|
+---+--------------------+
```

11.Додамо нові ідентифікатори до основного кадру даних

Код

```
customers = customers.join(customer_ids, 'id')
customers.show()
```

Вивід

```
+---+-------+-----------------+---------+--------------------+--------------------+
| id|max(id)|max(subscription)|max(year)|           interests|              new_id|
+---+-------+-----------------+---------+--------------------+--------------------+
|  1|      1|            19.99|     2023|Reading, Music, T...|5cafe5f8e6a0a21d4...|
|  6|      6|            24.99|     2023|Yoga, Meditation,...|5cd3bbaee6a0a21d4...|
|  3|      3|            29.99|     2024|Gaming, Theater, ...|5cd0ff8de6a0a21d4...|
|  5|      5|             12.0|     2024|Sports, Art, Trav...|5cc9c0e4e6a0a21d4...|
|  4|      4|             13.0|     2024|Dancing, Karaoke,...|5cbaf8e1e6a0a21d4...|
|  7|      7|            39.99|     2023|Swimming, Kayakin...|5caed1cbe6a0a21d4...|
|  2|      2|             9.99|     2023|Cooking, Photogra...|5cae9b3ce6a0a21d4...|
| 13|     13|            29.99|     2024|Rock Climbing, Sw...|5cd1db45e6a0a21d4...|
|  8|      8|            14.99|     2024|Painting, Sculptu...|5ce3e9a4e6a0a21d4...|
| 10|     10|            24.99|     2024|Painting, Sculptu...|5cb8c4b3e6a0a21d4...|
| 14|     14|            39.99|     2024|Surfing, Snowboar...|5cf3ba9de6a0a21d4...|
+---+-------+-----------------+---------+--------------------+--------------------+
```

12.Оберемо колонки, необхідні для БД MongoDB та перейменуємо їх.

Код

```
customers = customers.select('new_id', 'max(subscription)', 'max(year)',
'interests')
customers = customers.withColumnRenamed('new_id', '_id')
customers = customers.withColumnRenamed('max(subscription)', 'subscription')
customers = customers.withColumnRenamed('max(year)', 'year')
customers.show()
```

Вивід

```
+--------------------+------------+----+--------------------+
|                 _id|subscription|year|           interests|
+--------------------+------------+----+--------------------+
|5cafe5f8e6a0a21d4...|       19.99|2023|Reading, Music, T...|
|5cd3bbaee6a0a21d4...|       24.99|2023|Yoga, Meditation,...|
|5cd0ff8de6a0a21d4...|       29.99|2024|Gaming, Theater, ...|
|5cc9c0e4e6a0a21d4...|        12.0|2024|Sports, Art, Trav...|
|5cbaf8e1e6a0a21d4...|        13.0|2024|Dancing, Karaoke,...|
|5caed1cbe6a0a21d4...|       39.99|2023|Swimming, Kayakin...|
|5cae9b3ce6a0a21d4...|        9.99|2023|Cooking, Photogra...|
|5cd1db45e6a0a21d4...|       29.99|2024|Rock Climbing, Sw...|
|5ce3e9a4e6a0a21d4...|       14.99|2024|Painting, Sculptu...|
|5cb8c4b3e6a0a21d4...|       24.99|2024|Painting, Sculptu...|
|5cf3ba9de6a0a21d4...|       39.99|2024|Surfing, Snowboar...|
+--------------------+------------+----+--------------------+
```

13.Вивантажимо дані в MongoDB. На рис. 4.1. показано вивантажені дані.

Код

```
customers.write.format("mongodb").option("database",
"productdb").option("collection", "customers").mode("overwrite").save()
```

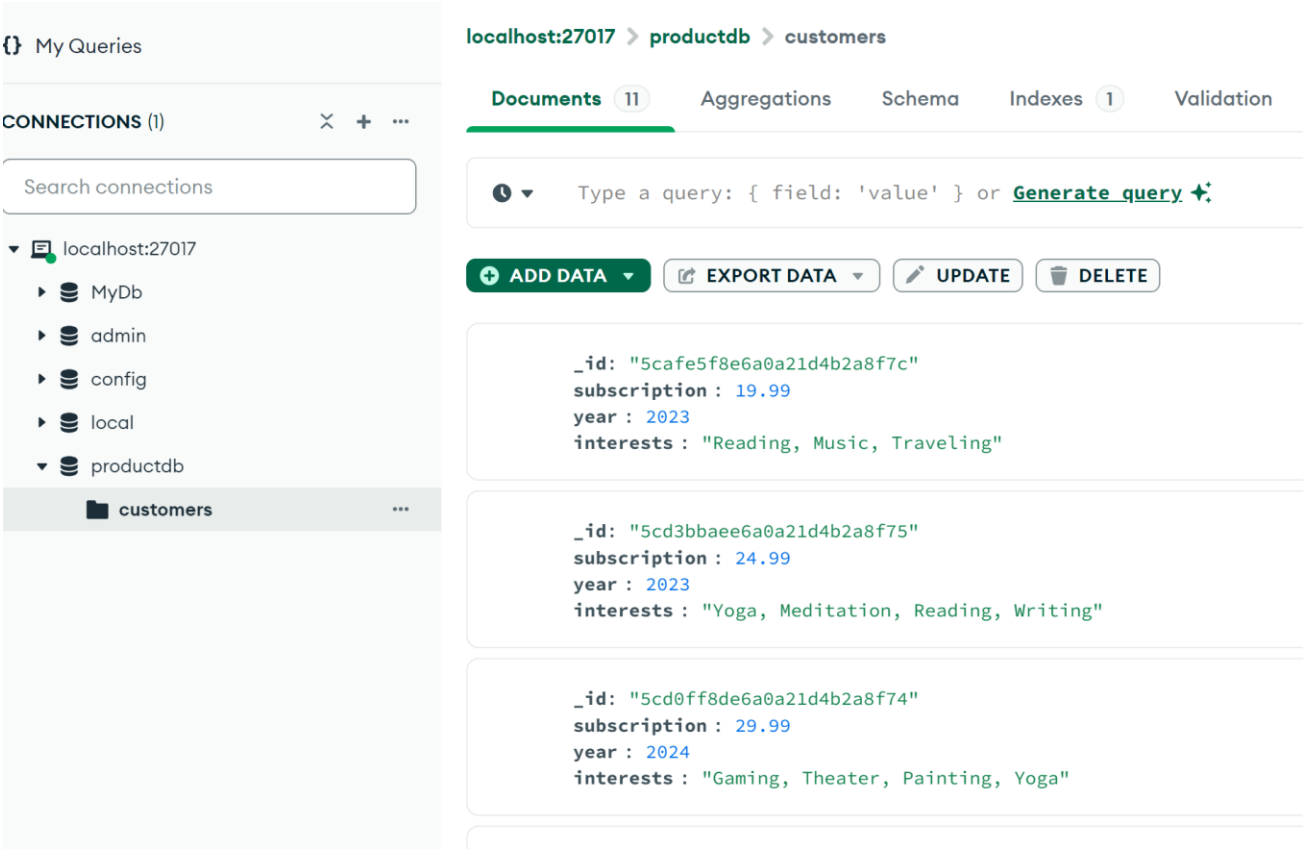

Рис. 4.1. Оброблені та вивантажені дані в MongoDB.

4.3. Індивідуальне завдання

Вашою задачею є здійснити подібне до прикладу перетворення для покупок. Для цього завантажте дані індивідуального варіанту за посиланням на порталі ДО.

Кожен крок завдання має супроводжуватись кодом та виводом 5 перших рядків кадру даних після перетворення.

1. Дані подано в файлах csv: purchases2021.csv, purchases2022.csv, purchases2023.csv, purchases2024.csv. Зчитати файли використовуючи PySpark. Об’єднати їх в один кадр даних, додавши колонку year (відповідно до назви файлу).
2. Як виявилось, дані не відповідають структурі основної колекції MongoDB та не можуть бути імпортовані як є. Знайти дублікати покупок (однакові id). Серед покупок з різною вартістю та однаковим id обрати максимальну.
3. Ідентифікатори користувачів, що використовувались в одному з офісів було спрощено та тепер вони не відповідають ідентифікаторам в основній БД. Скористайтесь файлом user_ids.csv для перетворення ідентифікаторів користувачів в правильний формат.

4. Також відгуки на покупки було винесено в окремий файл feedback.csv. В базі даних MongoDB відгук є полем колекції purchases. Перетворіть відгук таким чином.
5. Оберіть лише необхідні колонки. Перейменувати колонки аналогічно до прикладу обробки даних клієнтів.
6. Використовуючи MongoDB конектор вивантажте дані в MongoDB. Зробіть скріншот імпортованих даних в MongoDB Compass.
7. В PySpark визначте id 5 WIP клієнтів – клієнти, що купили продукцію на найбільшу суму.

Вимоги до оформлення звіту. Робота має бути представлена у вигляді звіту (файл pdf). Звіт має містити титульний лист, номер варіанта, по кожному запиту: скріншот середовища, код та вивід запиту.

Назва файлу: Прізвище_група

Роботи, виконані не за варіантом, не приймаються.

4.4. Контрольні питання

1. Що таке конвеєр ETL? Наведіть приклади задач, що можна вирішити за допомогою зазначеного конвеєру.
2. Яка різниця між ETL і ELT? Який підхід до яких задач є кращим?
3. Які операції виконуються в фазах витягування даних, перетворення та завантаження?
4. Як PySpark допомагає реалізувати конвеєр ETL?
5. Як PySpark можна підключити до БД MongoDB?
6. Назвіть функцію для перейменування колонок БД в PySpark.
7. За допомогою якої функції можна об’єднати рядки декількох таблиць?
8. Чому в MongoDB документи колекції слід зводити до однакового формату?

Оцінювання результатів навчання

Оцінювання практичних робіт

Кожна робота оцінюється за 100 бальною шкалою. Оцінювання відбувається строго у відповідності до пунктів індивідуального завдання, наведених в кожній практичній роботі, та пропорційно до ступеня повноти та коректності виконання кожного з них. Важливими вимогами до звіту з практичної роботи є:

1. Чіткий структурований опис проведеного дослідження, наведення необхідних графічних ілюстрацій.
2. Аналіз результатів, власні припущення та висновки студента, щодо отриманих закономірностей.
3. Кожна робота має обов'язково містити лістинг програмного коду, коректність якого та відповідність завданню оцінюється викладачем, програмний код студент має розуміти та виконати самостійно.
4. Роботи, виконані не за варіантом, не приймаються.

Вимоги до звіту

Звіт з виконання практичної роботи має містити:

- обкладинку в корпоративному стилі НТУ «Дніпровська політехніка» із зазначенням групи, ПІБ студента, викладача та теми роботи;
- номер варіанта, мета роботи, повний текст завдання;
- за кожним пунктом завдання опис, що було виконано, які результати отримано. Результати кожного пункту мають бути проаналізовані здобувачем;
- в кінці роботи має бути обов'язково наявний лістинг коду.

Звіт розміщується на сайті дистанційної освіти у відповідному розділі із завданням і оцінюється виходячи з 100 балів.

Рекомендовані джерела інформації

Базова:

1. Pedro Duarte Faria. Introduction to pyspark. – 2024. [Online] URL: https://pedropark99.github.io/Introd-pyspark/
2. Apache Spark documentation [Online]. URL: https://spark.apache.org/docs/latest/index.html
3. MongoDB documentation [Online]. URL: https://www.mongodb.com/docs/
4. Python MongoDB [Online]. URL: https://www.w3schools.com/python/python_mongodb_getstarted.asp
5. D. Paper Data Science Fundamentals for Python and MongoDB. Apress Media – 2018. – 201c.
6. Хабарлак К.С. Самонавчання складних систем [Електронний ресурс] : конспект лекцій для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / К.С. Хабарлак, Т.А. Желдак ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 112 с.

Додаткова:

1. K. Khabarlak "Tomato Plant Treatment Smart Suggestions Using Big Data Analytics" // Materials of the XII International Scientific-Practical Conference «Information Control Systems and Technologies», Odesa, Ukraine, 2024. - p. 214-216.
2. Practical Statistics for Data Scientists / P. Bruce, A. Bruce, P. Gedeck. – O'Reilly Media, 2020.

Навчальне видання

Хабарлак Костянтин Сергійович

АНАЛІЗ ТА ОБРОБКА ВЕЛИКИХ ДАНИХ

Методичні рекомендації до виконання практичних робіт
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Видано в авторській редакції.

Електронний ресурс.
Підписано до видання 01.10.2024. Авт. арк. 5,9.

Національний технічний університет «Дніпровська політехніка».
49005, м. Дніпро, просп. Дмитра Яворницького, 19.