**Anna Plust, 36148**

**Duże zbiory danych, 2022-23**

**Lab 5, Sprawozdanie**

**Napisanie aplikacji wczytującej dane z plików na dysku i uruchamianej lokalnie. (1, 2, 3, 4)**

public class Main {

    public static void main(String[] args) {

        SparkConf conf = new SparkConf()

                .setAppName("")

                .setMaster("local");

        JavaSparkContext sc = new JavaSparkContext(conf);

        SparkSession spark = SparkSession

                .builder()

                .sparkContext(sc.sc())

                .getOrCreate();

**// załadować pliki movies.csv oraz movies.json od oddzielnych obiektów Data Frame (typ Dataset<Row>)**

        Dataset<Row> moviesCsv = spark.read().csv("movies.csv");

        Dataset<Row> moviesJson = spark.read().json("movies.json");

**// wyświetlić ich zawartość (metoda: show()) oraz opis schematu (metoda: printSchema())**

        moviesCsv.show();

        moviesCsv.printSchema();

        moviesJson.show();

        moviesJson.printSchema();

**// utworzyć widok SQL dla jednego z utworzonych wcześniej obiektów DataFrame**

        moviesJson.createOrReplaceTempView("movies");

**// wysłać do widoku zapytanie wg schematu: SELECT \* FROM widok WHERE movieId > 500 ORDER BY movieId DESC**

        Dataset<Row> sqlDF = spark.sql("SELECT \* FROM movies WHERE movieId > 500 ORDER BY movieId DESC");

**//wyświetlić rezultat zapytania**

        sqlDF.show();

**// wczytać dane do kolekcji Dataset zwierającej obiekty typu Movie (czyli Dataset<Movie>)**

        Dataset<Movie> df = moviesJson.as(Encoders.bean(Movie.class));

**// wczytać plik z danymi o ocenach filmów (ratings.csv)**

        Dataset<Row> ratings = spark.read()

                .option("header", true)

                .option("inferSchema", true)

                .csv("ratings.csv");

**// wykonać operację grupowania ocen wg identyfikatora filmu oraz policzenia wystąpień w ramach grupy**

        Dataset x1 = ratings.groupBy(col("movieId")).count();

**// wykonać operację grupowania ocen wg identyfikatora filmu oraz policzenia średniej oceny w ramach grupy**

        Dataset x2 = ratings.groupBy(col("movieId")).avg("rating");

**// połączyć wyniki liczenia wystąpień i średniej wg identyfikatora filmu i wybranie tylko filmów ocenionych przynajmniej 5 razy**

        Dataset x3 = x1.join(x2, "movieId").where(col("count").geq(5));

**// połączyć wynik otrzymany w poprzednim podpunkcie ze zbiorem danych o filmach zawierającym tytuły**

        Dataset x4\_1 = moviesJson.select(col("movieId"), col("title"));

        Dataset x4 = x3.join(x4\_1, "movieId");  
**// wyświetlić tytuły filmów wraz z wartością średniej ocen posortowane wg jej wartości**

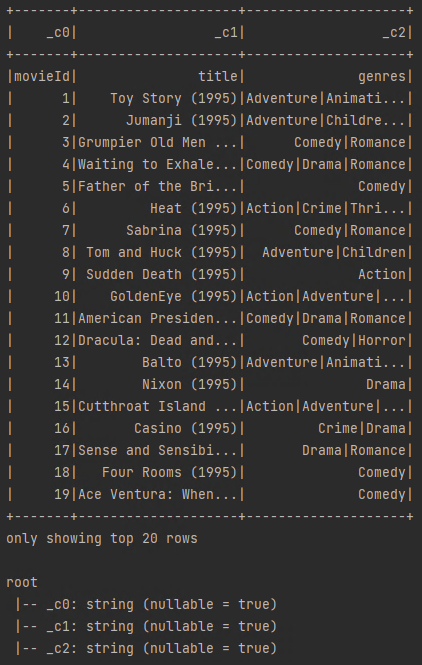
Dataset x5 = x4.select(col("movieId"), col("avg(rating)")).orderBy("avg(rating)");

x5.show();  
    }

}

**Aplikacja została uruchomiona w środowisku IntelliJ, a wyniki widoczne są następnej stronie.**

**1) Ładowanie obiektów Data Frame z plików tekstowych.  
Załadować pliki movies.csv oraz movies.json od oddzielnych obiektów Data Frame (typ Dataset<Row>). Wyświetlić ich zawartość (metoda: show()) oraz opis schematu (metoda: printSchema()).**

Plik CSV Plik JSON  
**** **A picture containing text, plaque, outdoor, screenshot

Description automatically generated**

**2) Zadawanie zapytań w języku SQL.**

**Utworzyć widok SQL dla jednego z utworzonych wcześniej obiektów DataFrame. Wysłać do widoku zapytanie wg schematu: SELECT \* FROM widok WHERE movieId > 500 ORDER BY movieId DESC. Wyświetlić rezultat zapytania.**

**A picture containing text, outdoor, plaque, screenshot

Description automatically generated**

**4) Operacje łączenia i agregacji.  
Wyświetlić tytuły filmów wraz z wartością średniej ocen posortowane wg jej wartości.**

**A picture containing text

Description automatically generated**

**Dostosowanie aplikacji i następnie uruchomienie jej w środowisku serwera. Wynik uzyskany w pkt. 4 zapisać w formacie CSV systemie plików hdfs klastra. (5, 6)**

public class Main {  
 public static void main(String[] args) {

**SparkConf conf = new SparkConf();**

JavaSparkContext sc = new JavaSparkContext(conf);

SparkSession spark = SparkSession  
 .builder()  
 .sparkContext(sc.sc())  
 .getOrCreate();

Properties prop = new Properties();

prop.setProperty("user", "bigdata");  
 prop.setProperty("password", "1234");

**// Wczytanie danych o filmach z serwera MySql**  
  **Dataset<Row> movies = spark.read().jdbc(  
 "jdbc:mysql://zecer.wi.zut.edu.pl:3306/bigdata",  
 "movies",  
 prop  
 );**

Dataset<Row> ratings = spark.read()  
 .option("header", true)  
 .option("inferSchema", true)  
 .csv("hdfs:///students/st36148/ratings.csv");

Dataset x1 = ratings.groupBy(col("movieId")).count();  
 Dataset x2 = ratings.groupBy(col("movieId")).avg("rating");  
 Dataset x3 = x1.join(x2, "movieId").where(col("count").geq(5));  
 Dataset x4\_1 = movies.select(col("movieId"), col("title"));  
 Dataset x4 = x3.join(x4\_1, "movieId");  
 Dataset x5 = x4.select(col("movieId"), col("avg(rating)")).orderBy(desc("avg(rating)"));

**// Wynik uzyskany w pkt. 4 zapisać w formacie CSV systemie plików hdfs klastra.**  
 **x5.write().csv("hdfs:/students/st36148/lab2-jdbc.csv");** }  
}

Po napisaniu aplikacji została ona spakowana przy pomocy komendy „package” w środowisku IntelliJ. Plik wynikowy został przesłany na serwer:

scp ./target/lab5-1.0-SNAPSHOT.jar [st36148@31.193.99.136:/home/st36148](mailto:st36148@31.193.99.136:/home/st36148)

Natępnie plik został przesłany na serwer hadoop:

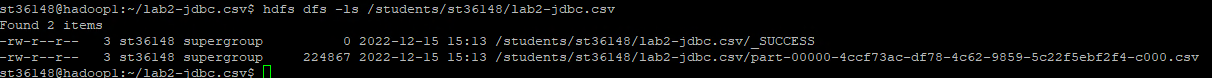
hdfs dfs -put lab5-1.0-SNAPSHOT.jar /students/st36148

Dodatkowo na serwer konieczne było wysłanie paczki *mysql-connector-java-8.0.29.jar*.

Aplikacja została uruchomina za pomocą komendy spark-submit z dołączonym connector’em mysql:

spark-submit --class Main --master spark://hadoop1:7077 --driver-class-path mysql-connector-java-8.0.29.jar --jars mysql-connector-java-8.0.29.jar lab5-1.0-SNAPSHOT.jar

**Zapisany plik na serwerze hdfs:**

****

**Zawartość pliku:**

**Graphical user interface, text

Description automatically generated**