FAKULTA ELEKTROTECHNICKÁ

České vysoké učení technické v Praze

B4M36DS2 – Database Systems 2

Practical Class 4

Perform MapReduce and Analytics with PySpark

Yuliia Prokop

prokoyul@fel.cvut.cz, Telegram **@Yulia_Prokop**

ČVUT FEL CourseWare Wiki https://cw.fel.cvut.cz/b241/courses/b4m36ds2/start

Run all tasks in Google Colab or Jupyter.

Add short explanations as text cells.

Exercise 0 - Install PySpark

What must be installed on your computer

If you use Google Colab:

- Python, Jupyter, and Pandas are already available. Just run:

```
pip install pyspark
```

(Mount Google Drive if your CSVs are there.)

- Access to the **CSV datasets** (movies.csv, reviews.csv, users.csv) and correct file paths.
- (Optional) **Google Drive integration** if running on Colab (to mount your Drive).

If you run locally (Python already installed):

- **Java Runtime Environment (JRE) or JDK 8+** (PySpark runs on the JVM)

```
pip install pyspark
```

(Install Pandas only if your environment doesn't have it.)

Exercise 1 - Start a SparkSession

Import the necessary libraries and start a SparkSession.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import desc, avg

spark =
SparkSession.builder.appName("DBS2_PySpark").getOrCreate()
```

Connect Google Drive if needed.

```
# Connect Google Drive (if needed)
from google.colab import drive
drive.mount('/content/drive')
```

Exercise 1 - Start a SparkSession

Update the paths to your files and load movies.csv, reviews.csv, and users.csv into Spark DataFrames.

```
movies =
spark.read.csv('/content/drive/MyDrive/CTU/DS22025/Dataset/movies.csv'
, header=True, inferSchema=True)

reviews =
spark.read.csv('/content/drive/MyDrive/CTU/DS22025/Dataset/reviews.csv
', header=True, inferSchema=True)

users =
spark.read.csv('/content/drive/MyDrive/CTU/DS22025/Dataset/users.csv',
header=True, inferSchema=True)
```

Exercise 2 - Preview the Data Structure

Display the first 5 rows of each DataFrame (movies, reviews, users).

```
movies.show(5)
reviews.show(5)
users.show(5)
```

Exercise 3 - Analyze via DataFrame API

1. Show the Top 10 movies by number of reviews (DataFrame API).

```
# Task 1. Top-10 movies by number of reviews
cnt = reviews.groupBy("movie_id").count()

top_movies = cnt.join(movies, cnt.movie_id == movies.id)\
.select("title", "count") \
.orderBy(desc("count"))
top_movies.show(10)
```

Exercise 3 - Analyze via DataFrame API

2. Compute the average movie rating by year (DataFrame API).

```
# Task 2. Average movie rating by year
reviews_with_movie = reviews.join(movies, reviews.movie_id
== movies.id)

avg_rating_by_year = reviews_with_movie.groupBy("year") \
.agg(avg("rating").alias("avg_rating")) \
.orderBy("year")
avg_rating_by_year.show()
```

Exercise 4 - MapReduce via RDD API (WordCount)

WordCount: Most Frequent Words in Reviews

Count how many times each word appears in all reviews.

Show the top 10 most frequent words.

Implement an **explicit MapReduce pattern** with RDDs:

- flatMap/map (map),
- reduceByKey (combiner + reduce),
- take top-N.

Exercise 4 - Solution

```
# Extract a single text column and convert DF -> RDD[Row];
# emit exactly one string per row
# (empty string for missing/nulls to keep downstream logic simple)
review_text_rdd = reviews.select("review_text").rdd.flatMap(lambda
row: [row.review_text if row.review_text else ""])

# Split each line into tokens; flatMap flattens lists of words into a single RDD stream
words = review_text_rdd.flatMap(lambda line: line.split())

# Normalize tokens: lowercase + trim leading/trailing punctuation (strip affects
only edges)
words = words.map(lambda w: w.lower().strip(",.!?;:-\"'"))

# MAP: Emit (word, 1) pairs
word_pairs = words.map(lambda word: (word, 1))

# Sum counts by key;
# reduceByKey performs local combining on each partition before the shuffle
word_counts = word_pairs.reduceByKey(lambda a, b: a + b)

# Take Top-20 by frequency without a full global sort
for word, count in word_counts.takeOrdered(20, key=lambda x: -x[1]):
  print(word, count)
```

Exercise 5

Review Count per Movie

For each movie, count the total number of reviews.

Show the top 10 movies by number of reviews (display movie titles).

Exercise 5 - Solution

```
# MAP: For each review, create a pair (movie_id, 1)
movie_pairs = reviews.rdd.map(lambda row: (row['movie_id'], 1))

# REDUCE: Sum the counts for each movie_id
movie_review_counts = movie_pairs.reduceByKey(lambda a, b: a + b)

# Convert to DataFrame and join with movies for movie titles (optional)
movie_review_counts_df = movie_review_counts.toDF(["movie_id",
"review_count"])

result = movie_review_counts_df.join(movies,
movie_review_counts_df.movie_id == movies.id) \
    .select(movies.title, "review_count") \
    .orderBy("review_count", ascending=False)

result.show(10)
```

Exercise 6

Review Count per User

For each user, count the total number of reviews they have written.

Show the top 10 most active users (display user names).

Exercise 6 - Solution

```
# MAP: For each review, create a pair (user_id, 1)
user_pairs = reviews.rdd.map(lambda row: (row['user_id'], 1))


# REDUCE: Sum the counts for each user_id
user_review_counts = user_pairs.reduceByKey(lambda a, b: a + b)


# Convert to DataFrame and join with users for user names (optional)
user_review_counts_df = user_review_counts.toDF(["user_id",
"review_count"])
result = user_review_counts_df.join(users,
user_review_counts_df.user_id == users.id) \
.select(users.name, "review_count") \
.orderBy("review_count", ascending=False)


result.show(10)
```

Compute Average Rating per Movie (RDD)

For each movie, calculate the average rating from all its reviews.

Show the top 10 highest-rated movies (with titles)

Exercise 7

```
# MAP: For each review, create a pair (movie_id, (rating, 1))
movie_rating_pairs = reviews.rdd.map(lambda row: (row['movie_id'],
(row['rating'], 1)))

# REDUCE: Aggregate sum of ratings and count for each movie_id
rating_totals = movie_rating_pairs.reduceByKey(lambda a, b: (a[0] +
b[0], a[1] + b[1]))

# MAP: Compute average = sum / count for each movie_id
avg_rating = rating_totals.mapValues(lambda x: x[0] / x[1])

# Convert to DataFrame and join with movies for movie titles (optional)
avg_rating_df = avg_rating.toDF(["movie_id", "avg_rating"])
avg_rating_df.join(movies, avg_rating_df.movie_id == movies.id) \
.select(movies.title, "avg_rating") \
.orderBy("avg_rating", ascending=False) \
.show(10)
```

Tasks 1-6 – Self-study

1. **Average Rating per User:** For each user, calculate their average review rating. Show the top 10 users with the highest average rating (display user names).
2. **Most Popular Genre by Review Count:** For each genre (split the genres field if there are multiple), count the total number of reviews for movies in that genre. Show the most reviewed genre.
3. **Movie with the Most Positive Reviews:** For each movie, count the number of reviews where sentiment is "positive". Show the film with the most positive reviews.
4. **User Who Reviewed the Widest Variety of Genres:** For each user, count how many different genres they have reviewed across all their reviews. Show the user who reviewed the most unique genres.
5. **Most Controversial Movie:** For each movie, calculate the standard deviation of all its ratings. Show the top 5 movies with the highest rating standard deviation (i.e., most controversial).
6. **Friend Pairs: Most Reviews of the Same Movies:** For each pair of users who are friends (use the friends field in users), count how many movies both have reviewed. Show the friend pair(s) with the most shared reviewed movies.

Submission Instructions

- Submit a **single PDF exported from Google Colab** that includes **all code cells and their outputs**.
- **Run every cell** in order before exporting. No empty outputs.
- If randomness is involved, **set a seed** so results are reproducible.
- Long outputs: show the **top 20 / head** unless the task explicitly requires full output.
- In Colab: **File → Print** → in the browser print dialog, choose **Save as PDF**.
  (This preserves code + outputs exactly as seen.)
- Share the source notebook (.ipynb) with **"Anyone with the link can view."**
- Include the view link on the first page of your PDF.
- Submit the PDF to the BRUTE system (PR4)