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

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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
```

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

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

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

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

```
+--------------------+--------------------+----+--------------------+------------+------------
|                  id|               title|year|              genres|director_ids|
+--------------------+--------------------+----+--------------------+------------+------------
|9883a8b0-7d12-44e...|              Avatar|2009|Action|Adventure|...|        NULL|
|751e58b9-dd04-4b9...|Pirates of the Ca...|2007|Adventure|Fantasy...|       d1704|
|72b81637-fa68-46b...|             Spectre|2015|Action|Adventure|...|        NULL|
|0d1d9a1e-5640-46d...|The Dark Knight R...|2012|Action|Crime|Dram...|        NULL|
|24119185-79fe-410...|         John Carter|2012|Action|Adventure|...|          d7|a60900|a2104
+--------------------+--------------------+----+--------------------+------------+------------
only showing top 5 rows

+--------------------+--------------------+--------------------+------+-----------+-----------
|                  id|            movie_id|             user_id|rating|review_text|
+--------------------+--------------------+--------------------+------+-----------+-----------
|c1be78b5-4f47-4ae...|1fe19dc8-48ae-401...|ee70fa2f-ad03-468...|   9.9|       Meh.|"[{""user_i
|4c69c546-896f-45c...|1b1c46be-7f24-4ec...|77de6356-3997-49b...|   9.0| Bad movie.|
|86d8ed20-46fc-4ea...|51744bcd-2c89-453...|8ece220e-afe4-4a7...|   5.9|     Great!|"[{""user_i
|7d138db5-47b5-4f3...|3682cc40-e6f5-4a1...|fa115698-5061-453...|   8.6|     Great!|
|69eff528-7022-408...|487c5b50-77d0-446...|4c7f9f10-e09a-4cf...|   2.9|     Great!|"[{""user_i
+--------------------+--------------------+--------------------+------+-----------+-----------
only showing top 5 rows

+--------------------+------+--------------------+--------------------+--------------------+--
|                  id|  name|   registration_date|    favourite_movies|             friends|pr
+--------------------+------+--------------------+--------------------+--------------------+--
|f9315b26-4101-458...|User_1|2015-08-01 10:33:...|['6077c79b-e6bf-4...|['eeae3048-7d60-4...|
|b368dcb8-e2dd-4b0...|User_2|2017-03-17 10:33:...|['b326ecfc-530e-4...|['410cc633-ee84-4...|
|5552b918-5609-4be...|User_3|2016-06-21 10:33:...|['82a15444-94e4-4...|['d1c8561e-3b7c-4...|
|ff0bb973-2c47-4f2...|User_4|2016-08-20 10:33:...|['67c546ab-427c-4...|['3c13da14-624a-4...|
|62f8db9c-ed57-4a3...|User_5|2015-11-03 10:33:...|['e4d76168-8d21-4...|['7c3fd02d-bec4-4...|
+--------------------+------+--------------------+--------------------+--------------------+--
only showing top 5 rows
```

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

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

```
+--------------------+-----+
|               title|count|
+--------------------+-----+
```

```
|                Dick|   46|
|            Rounders|   45|
|         The Postman|   44|
|       The Blue Bird|   43|
|           Unnatural|   42|
|Get Him to the Greek|   42|
|X-Men: The Last S...|   41|
|Spy Kids 3-D: Gam...|   41|
|             Control|   41|
|The Velocity of Gary|   41|
+--------------------+-----+
only showing top 10 rows
```

```
# 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()
```

```
+----+------------------+
|year|        avg_rating|
+----+------------------+
|NULL| 5.254545454545454|
|1916|  5.29090909090909|
|1925| 4.923684210526316|
|1927| 5.404347826086957|
|1929| 5.308333333333333|
|1930|  5.04090909090909|
|1932| 4.930000000000001|
|1933| 5.853061224489796|
|1934| 4.806666666666667|
|1935| 6.066666666666666|
|1936|5.5218181818181815|
|1937| 5.836734693877551|
|1938| 5.242000000000001|
|1939| 5.346478873239437|
|1940| 5.354304635761588|
|1941| 5.142857142857144|
|1942| 5.078431372549018|
|1944| 5.638888888888889|
|1945| 5.855445544554455|
|1946| 5.449438202247191|
+----+------------------+
only showing top 20 rows
```

```
# MAP: Create an RDD of review_text strings (handle missing/nulls)
review_text_rdd = reviews.select("review_text").rdd.flatMap(lambda row: [row.review_text if r
print(review_text_rdd.take(5))

# MAP: Split each line into words
words = review_text_rdd.flatMap(lambda line: line.split())
print(words.take(5))

# MAP: Normalize to lowercase, strip punctuation
words = words.map(lambda w: w.lower().strip(",.!?;:-\"'"))
print(words.take(5))

# MAP: Create (word, 1) pairs
word_pairs = words.map(lambda word: (word, 1))
print(word_pairs.take(5))

# REDUCE: Aggregate (sum) counts for each word
word_counts = word_pairs.reduceByKey(lambda a, b: a + b)
print(word_counts.take(5))

# Show top 20 most frequent words
```

```
for word, count in word_counts.takeOrdered(20, key=lambda x: -x[1]):
  print(word, count)
```

```
['Meh.', 'Bad movie.', 'Great!', 'Great!', 'Great!']
['Meh.', 'Bad', 'movie.', 'Great!', 'Great!']
['meh', 'bad', 'movie', 'great', 'great']
[('meh', 1), ('bad', 1), ('movie', 1), ('great', 1), ('great', 1)]
[('legendary', 30197), ('meh', 29792), ('bad', 29843), ('movie', 29843), ('great', 30168)]
legendary 30197
great 30168
bad 29843
movie 29843
meh 29792
```

```
# 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)
```

```
+--------------------+------------+
|               title|review_count|
+--------------------+------------+
|                Dick|          46|
|            Rounders|          45|
|         The Postman|          44|
|       The Blue Bird|          43|
|Get Him to the Greek|          42|
|           Unnatural|          42|
|Spy Kids 3-D: Gam...|          41|
|The Velocity of Gary|          41|
|             Control|          41|
|X-Men: The Last S...|          41|
+--------------------+------------+
only showing top 10 rows
```

```
# 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)
```

```
+----------+------------+
|      name|review_count|
+----------+------------+
|User_27403|          14|
| User_8647|          14|
|  User_666|          14|
|User_19535|          13|
|User_28315|          13|
|User_25610|          13|
| User_1393|          13|
|User_17629|          13|
```

```
|User_27783|          12|
|User_17230|          12|
+----------+------------+
only showing top 10 rows
```

```
# 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)
```

```
+--------------------+------------------+
|               title|        avg_rating|
+--------------------+------------------+
|        The Terminal| 7.842857142857143|
|      Dracula Untold|7.3478260869565215|
|       Party Monster| 7.265000000000001|
|         The Dilemma| 7.235714285714286|
|Live Free or Die ...| 7.152941176470588|
|Good Night, and G...| 7.083870967741935|
|             Jason X| 7.056521739130435|
|Smilla's Sense of...| 7.023529411764706|
|           Soul Food| 7.018518518518518|
|             Ant-Man| 7.014285714285715|
+--------------------+------------------+
only showing top 10 rows
```