Execution Steps for PySpark Code and Test Case

To execute the provided PySpark code and test case, follow the steps below:

1. Set Up Your Environment:

Ensure you have the necessary tools installed and set up before running the PySpark code and the associated test case.

a. Install PySpark:

Install PySpark Python package to interact with Spark:

```
pip install pyspark
```

b. Install Additional Dependencies (requests, gzip, shutil):

Ensure that the required libraries such as requests, gzip, and shutil are installed.

```
pip install requests
```

(Note: gzip and shutil are built-in Python libraries, so they do not require installation.)

2. Prepare the IMDb Dataset URLs:

The code provided downloads three IMDb datasets:

- **title.ratings.tsv.gz** (Ratings of movies)
- **title.basics.tsv.gz** (Movie details)
- **title.principals.tsv.gz** (Credits/Persons related to movies)

3. Execute the PySpark Code:

a. Download and Extract IMDb Datasets:

- The function download_and_extract_imdb_data will download the required datasets and extract them if they are in .gz compressed format.
    - This will happen automatically when the code is run.
    - The downloaded and extracted files will be stored in the directory imdb_datasets or test_imdb_datasets based on the test case.

b. Load and Process the Data:

- PySpark will load the datasets (ratings_df, movies_df, principals_df) into DataFrames with the defined schemas.
- Various transformations are applied:
    - Filter movies with a minimum of 500 votes.

    - Calculate a ranking score for movies based on votes and average ratings.
    - Join the filtered movie dataset with the rating and movie basic details.

c. Display Results:

- The results are displayed using show() method:
    - **Top 10 movies with ranking score**: This will be displayed in a tabular format showing movie titles and their ranking score.
    - **Top 10 movies with titles based on ranking score**: Display titles of top 10 movies.
    - **Most credited persons for top 10 movies**: This will display the credited persons for the top movies based on the number of movie credits.

4. Execute the Test Case:

a. Set Up Test Case Environment:

1. **Initialize Spark Session**: The test case initializes a Spark session using SparkSession.builder.
2. **Download and Extract Data**: The datasets will be downloaded and extracted as part of the test case.
3. **Load the Data**: Data is loaded into DataFrames with appropriate schemas.
4. **Test the Ranking Logic**: In the test_ranking_logic test case, it:
    - Filters movies with 500 or more votes.
    - Calculates a ranking score based on votes and average rating.
    - Joins with the movie dataset and selects top 10 movies.
    - Asserts that exactly 10 movies are returned.
    - Displays the movie titles using the show() method.
5. **Test Credited Persons**: In the test_credited_persons test case, it:
    - Filters for the top 10 movies listed in previous step.
    - Fetches the persons credited for the movies.
    - Asserts that there are credited persons and displays the list.

b. Run the Test Case:

To run the test case, simply execute the following:

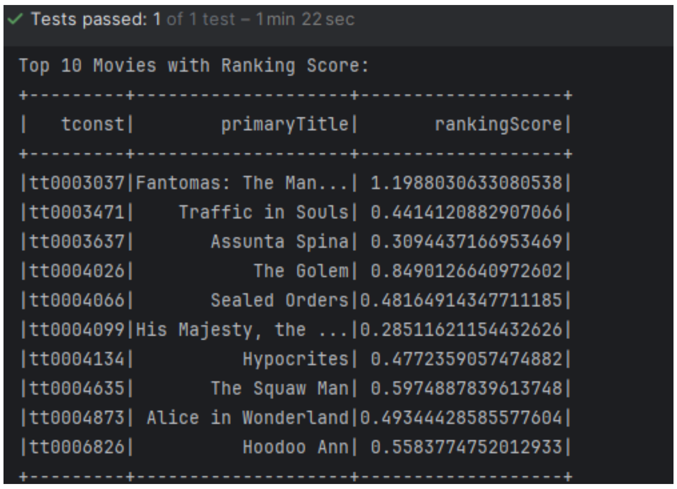
```
python  unittest <filename>.py
```

This will trigger the unittest framework, execute the tests, and display the results in the terminal.

5. Verify the Output:

- During the test case execution, the following will be displayed in the terminal:
    - **Top Movie with ranking score**: The list of top 10 movie with their ranking score.
    - **Top Movie Titles**: The list of top 10 movie titles based on the ranking score.
    - **Most Credited Persons**: The list of persons credited the most for the above top 10 movies.

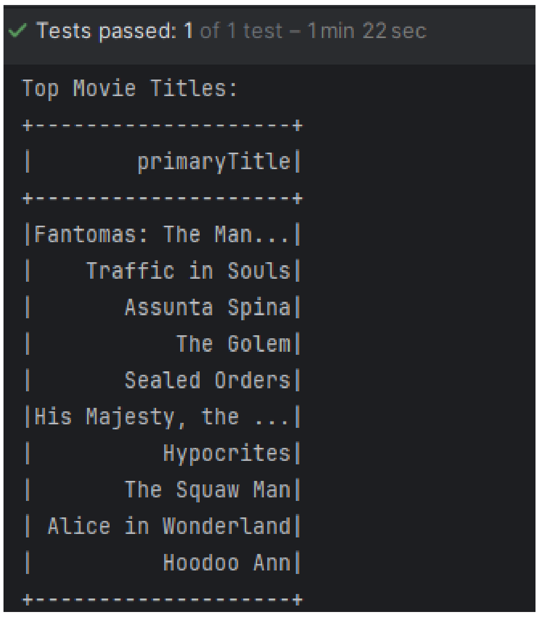
Results output for **Top Movies with ranking score**:



```
✓ Tests passed: 1 of 1 test – 1 min 22 sec

Top 10 Movies with Ranking Score:
+---------+--------------------+-------------------+
|   tconst|        primaryTitle|       rankingScore|
+---------+--------------------+-------------------+
|tt0003037|Fantomas: The Man...| 1.1988030633080538|
|tt0003471|    Traffic in Souls| 0.4414120882907066|
|tt0003637|       Assunta Spina| 0.3094437166953469|
|tt0004026|           The Golem| 0.8490126640972602|
|tt0004066|       Sealed Orders|0.48164914347711185|
|tt0004099|His Majesty, the ...|0.28511621154432626|
|tt0004134|          Hypocrites| 0.4772359057474882|
|tt0004635|       The Squaw Man| 0.5974887839613748|
|tt0004873| Alice in Wonderland|0.49344428585577604|
|tt0006826|          Hoodoo Ann| 0.5583774752012933|
+---------+--------------------+-------------------+
```

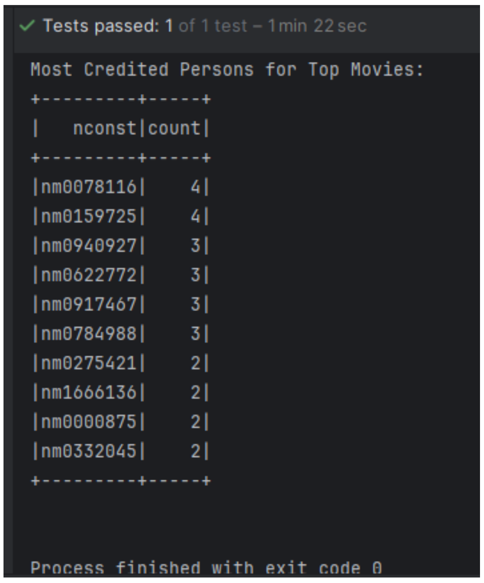
Results output for the **Top Movie Titles based on ranking score**:



```
✓ Tests passed: 1 of 1 test – 1 min 22 sec

Top Movie Titles:
+--------------------+
|        primaryTitle|
+--------------------+
|Fantomas: The Man...|
|    Traffic in Souls|
|       Assunta Spina|
|           The Golem|
|       Sealed Orders|
|His Majesty, the ...|
|          Hypocrites|
|       The Squaw Man|
| Alice in Wonderland|
|          Hoodoo Ann|
+--------------------+
```

Results output for the **Most Credited Persons**



```
✓ Tests passed: 1 of 1 test – 1 min 22 sec

Most Credited Persons for Top Movies:
+---------+-----+
|   nconst|count|
+---------+-----+
|nm0078116|    4|
|nm0159725|    4|
|nm0940927|    3|
|nm0622772|    3|
|nm0917467|    3|
|nm0784988|    3|
|nm0275421|    2|
|nm1666136|    2|
|nm0000875|    2|
|nm0332045|    2|
+---------+-----+


Process finished with exit code 0
```

6. Clean Up:

a. Spark Session Termination:

- After execution, the Spark session is stopped by calling spark.stop().

b. Remove Downloaded Files (Optional):

- You may want to remove the downloaded and extracted files from the imdb_datasets or test_imdb_datasets directory to free up space.

Optimization of PySpark Code:

There are the optimizations made to your PySpark code and test case

1. **Use of Broadcast Join:**
    - Broadcasting the movies_df DataFrame in the ranked_movies.join to avoid shuffling of the smaller dataset (movies_df).
2. **Avoid Collecting Top Movie IDs into Driver Memory:**
    - Instead of collecting the top movie IDs with collect() and filtering them, I used a join with principals_df to directly fetch credited persons for the top movies. This avoids unnecessary memory overhead on the driver.
3. **Partitioning and Cache Optimizations:**
    - Used .cache() on large DataFrames like ratings_df, movies_df, and principals_df to avoid repeated reads from disk, and make the code more efficient when performing multiple operations on them.

4. **Efficiency in Reading CSV Files:**
    - The file reading is done only once per dataset instead of multiple read operations. Loaded the static datasets (movies_df and credits_df) only once and reused them throughout the application, avoiding repeated reads and computations.
5. **Reduced Shuffle Partitions**:
    - Set spark.sql.shuffle.partitions = 8 for local testing, reducing the number of partitions during shuffle operations. This decreases processing overhead for small-scale local runs.
6. **Adjusted Memory Allocation**:
    - Configured spark.executor.memory to allocate 2GB for Spark executors, improving memory handling for larger datasets.
7. **Avoid Redundant Computations**:
    - Pre-calculated average_num_votes once using ratings_df outside the streaming process. This avoids recalculating the same value for every batch.
8. **Selective Column Loading**:
    - Only loaded and processed the required columns from each dataset (tconst, primaryTitle, numVotes, averageRating, etc.), reducing memory usage and improving performance.
9. **File Error Handling**:
    - Try except is used to handle exceptions gracefully and either handle the error or print out a useful error message
10. **Limit Rows for Top 10 Results**:
    - Applied .limit(10) after sorting the ranked movies, ensuring that only the top 10 rows are processed further. This reduces data volume and computational cost downstream.
11. **Aggregation Optimization**:
    - Grouped and aggregated top_movies_credits using only necessary columns to find the most credited persons, minimizing processing on unnecessary data.

Optimized Test Case:

- **Parallel Data Loading:**
    - Spark's ability to run operations in parallel can be fully utilized by controlling the number of partitions and parallelizing operations.
- **Avoid Collecting Data to Driver:**

- Avoid unnecessary use of collect() which gathers data into the driver. The join operation between principals_df and top_movies ensures that the data is processed efficiently on the worker nodes.

Things to care:

1. **Recompute ranked_movies_with_titles**:

   Ensure this DataFrame is consistent with the ranking calculation and includes only the top 10 ranked movies with proper titles.

2. **Use collect() only when absolutely necessary**:

   Avoid excessive use of collect() as it materializes the DataFrame into memory and can disrupt subsequent Spark operations.

3. **Directly display top_movies for better clarity**:

   Call show() on the top_movies DataFrame, which is already calculated and represents the top-ranked movies with titles.

4. **Get the most credited persons for the top 10 movies:** You already have a mechanism to calculate the ranking score and retrieve the top 10 movies. Make sure we join this with the principals_df to get the credited persons so that it will reflect most credited persons for those top 10 movies which are already computed.