CSE488: PySpark Exercise using RDD

1. **Create and Transform an RDD**:

- Create an RDD from a list of integers.
- Perform the following:
  - Multiply each number by 3 using map.
  - Filter the numbers greater than 10 using filter.
  - Collect and display the final result.

2. **Read a Text File**:

- Use textFile to read a file containing sentences (e.g., "hello world", "spark is great").
- Perform the following:
  - Split each line into words using flatMap.
  - Count the total number of words using count

3. **GroupByKey and ReduceByKey**:

- Create an RDD with pairs of student names and their scores, e.g., `[("Alice", 85), ("Bob", 90), ("Alice", 95)]`.
- Use:
  - `groupByKey` to group scores by student.
  - `reduceByKey` to calculate the total score for each student.

4. **RDD Persistence**:

- Create an RDD from a large list (simulate it by generating numbers from 1 to 1,000,000).
- Perform multiple actions (e.g., `count`, `sum`) without caching and measure execution time.
- Repeat with `cache` or `persist` and compare the performance.

5. Custom Transformations:

- Create an RDD of numbers.
- Write a custom transformation to identify and filter out prime numbers.

6. **Transformation and Action Workflow**:

- Load a dataset (e.g., CSV or text file) containing the following:

```
Product,Category,Price
Laptop,Electronics,800
Shoes,Clothing,50
Phone,Electronics,500
```

- Perform the following:
    - Filter products with a price greater than 100.
    - Map to get only the product names.
    - Count the number of products in each category using `map` and `reduceByKey`.

7. **Integration with Spark SQL**:

- Load a JSON dataset containing information about students:

```
[{"name": "Alice", "age": 20, "grade": "A"},
 {"name": "Bob", "age": 22, "grade": "B"}]
```

- Perform the following:
    - Register the data as a temporary SQL table.
    - Query students who have a grade "A" using Spark SQL.
    - Save the result to a new JSON file.

8. **Advanced Word Count with Sorting**:

- Extend the word count program to:
    - Sort words by their frequency in descending order.
    - Display the top 5 most frequent words.

9. **Custom Aggregations with `aggregateByKey`**:

- Create an RDD with pairs of cities and temperatures, e.g., `[("NY", 32), ("LA", 75), ("NY", 28)]`.
- Use **`aggregateByKey`** to calculate the average temperature for each city.