**Ansh Ranjan  
Databricks Case Study  
EXERCISE 5 – Advanced Spark Topics and Optimization**

**TASK 1: Implement caching and Broadcasting**

1. Caching a dataframe in memory allows for faster reuse
2. To cache a dataframe run df.cache()  
   A black background with white text

   AI-generated content may be incorrect.
3. If we are joining a large dataframe with a smaller one. We can broadcast the smaller dataframe across all worker nodes to avoid shuffling.  
   A screen shot of a computer code

   AI-generated content may be incorrect.

**TASK 2: Use spark’s advanced concepts**

**1. Use DataFrames and Spark SQL (not RDDs)**

* DataFrames are optimized by **Catalyst** and **Tungsten**, making them much faster.
* Avoid low-level RDDs unless absolutely necessary.

**2. Cache / Persist Smartly**

* Use .cache() or .persist() **only** when you reuse a DataFrame **multiple times**.
* Don’t over-cache — memory is limited!

**3. Broadcast Small Lookup Tables**

* When joining with small datasets (< 100 MB), use:

from pyspark.sql.functions import broadcast  
df\_large.join(broadcast(df\_small), "key")

**4. Filter Early, Select Only Needed Columns**

* Apply .filter() and .select() early to reduce data volume.

df.select("col1", "col2").filter("col1 > 100")

**5. Partition Smartly**

* Use .repartition() to distribute load for wide transformations or large joins.
* Use .coalesce(1) **only** for final small outputs (e.g., exports).

df = df.repartition(10, "some\_column")

**6. Use Delta Lake Format**

* Delta tables are faster for reads, support ACID, schema evolution, and time travel.
* Use OPTIMIZE and ZORDER to improve read performance.

OPTIMIZE my\_table ZORDER BY (col\_name)

**7. Avoid Exploding Joins & Shuffles**

* Shuffles are expensive (joins, groupBy, distinct).
* Repartition wisely to avoid skew.

**8. Monitor with Spark UI**

* Always check stages and tasks in **Spark UI** (or the **Databricks Job Run view**).
* Look out for:
  + Long tasks
  + Skewed stages
  + Too many small files