**🔹 Lab 1: Spark Architecture – Driver, Executor, Shuffle**

**Objective:**  
Understand the physical architecture of Spark execution and identify shuffle boundaries in transformations.

**Dataset:**

* products.csv (id, name, category, price)
* orders.csv (order\_id, product\_id, quantity, region)

**Tasks:**

1. Load both datasets as DataFrames.
2. Join products and orders, group by category, and calculate total revenue.
3. Open Spark UI and identify:
   * Number of stages
   * Where shuffle occurred
4. Discuss the role of Driver vs Executor in this job.
5. Suggest how shuffle can be minimized for similar workloads.

**Expected Output:**  
A summary table of revenue per category and a screenshot/observation from Spark UI stages.

**🔹 Lab 2: DataFrame API Essentials**

**Objective:**  
Practice using DataFrame APIs for filtering, transformation, aggregation, and column operations.

**Dataset:**

* users.csv (id, name, age, region)
* activity.csv (user\_id, date, activity\_type, duration\_min)

**Tasks:**

1. Load both datasets.
2. Filter users age > 30, and join with activity logs.
3. For each user, calculate total and average activity duration.
4. Use .selectExpr() to create a new column: duration\_hrs = duration\_min / 60.
5. Save final result as a partitioned Parquet file by region.

**Expected Output:**  
Partitioned DataFrame with user activity stats and newly derived columns.

**🔹 Lab 3: Cache vs Checkpoint**

**Objective:**  
Understand when to use .cache() vs .checkpoint() in long pipelines.

**Dataset:**  
Use output from Lab 2 (user activity stats).

**Tasks:**

1. Create a long pipeline: read → join → filter → groupBy → sort.
2. Cache at the stage before sort and observe DAG/stages.
3. Repeat with checkpoint at the same stage.
4. Compare lineage lengths and stage duration from Spark UI.

**Expected Output:**  
DAG graphs and comparison of performance and lineage depth between caching and checkpointing.

**🔹 Lab 4: Spark UI Deep-Dive**

**Objective:**  
Use Spark Web UI to analyze performance metrics and troubleshoot slow stages.

**Tasks:**

1. Reuse any pipeline from Labs 1–3.
2. Open Spark UI during job execution.
3. Capture:
   * Number of tasks per stage
   * Shuffle Read/Write metrics
   * Execution time per task
4. Identify slowest stage and reason.

**Expected Output:**  
Screenshot of Spark UI with annotations and notes explaining performance bottlenecks.

**🔹 Lab 5: RDD vs DataFrame**

**Objective:**  
Compare performance and readability of RDD-based transformations with DataFrame API.

**Dataset:**

* clicks.csv (user\_id, page, timestamp)

**Tasks:**

1. Load the dataset as RDD and as DataFrame.
2. For both versions, calculate clicks per user.
3. Time the operations using time.time() and compare execution plans with .explain().
4. Identify which is better and why.

**Expected Output:**  
Execution time comparison and rationale for preferring DF API in production.

**🔹 Lab 6: Parquet Partition Discovery**

**Objective:**  
Understand how Spark reads only required partitions from Parquet.

**Dataset:**

* /data/campaign\_data/year=2023/month=06/\*.parquet

**Tasks:**

1. Load the Parquet folder using spark.read.parquet()
2. Filter data for month=06
3. Use .explain(True) to confirm partition pruning
4. Compare scan time with unpartitioned version

**Expected Output:**  
Evidence that Spark scans only the necessary partition(s).

**🔹 Lab 7: Broadcast Hints for Joins**

**Objective:**  
Reduce shuffle using broadcast join hints.

**Dataset:**

* transactions.csv (txn\_id, user\_id, amount)
* users.csv (user\_id, name, region) *(assume small enough to broadcast)*

**Tasks:**

1. Load both datasets
2. Perform join normally and record stage count and shuffle size
3. Apply broadcast() hint to users and rerun
4. Observe stage plan and performance difference

**Expected Output:**  
Fewer shuffle reads/writes and improved join performance with broadcast.

**🔹 Lab 8: External Source Read (Blob / ADLS)**

**Objective:**  
Read partitioned data from cloud storage and apply partition pruning.

**Dataset:**

* ADLS/Blob: /mnt/blob/payments/year=2024/month=03/\*.parquet

**Tasks:**

1. Mount or configure ADLS/Blob access using Spark configs
2. Read the Parquet dataset
3. Filter for a specific month and confirm pruning with .explain(True)
4. Show schema and confirm that partition columns are available

**Expected Output:**  
Correct data read with pruning, verified via physical plan.

**🔹 Optional Stretch Lab: Advanced UDF Profiling**

**Objective:**  
Compare Spark UDF with pandas UDF for row-wise transformations.

**Dataset:**

* transactions.csv (txn\_id, user\_id, amount, risk\_flag)

**Tasks:**

1. Write a Spark UDF to assign risk scores
2. Rewrite the logic using pandas\_udf
3. Time both operations and monitor stage performance in Spark UI
4. Analyze DAG for serialization bottlenecks

**Expected Output:**  
Execution time delta and stage performance insight, with pandas UDF being faster.